

August 1977

**Application Techniques
for the MCS-48[™] Family**

Lionel Smith
Microcomputer Applications

Related Intel Publications

“MCS-48™ Microcomputer User’s Manual”

“Using the 8251 Universal Synchronous/Asynchronous Receiver/Transmitter”

“8255 Programmable Peripheral Interface Applications”

Application Techniques for the MCS-48™ Family

Contents

INTRODUCTION	1
THE MCS-48™ FAMILY	1
ANALOG I/O	3
TABLE LOOKUP TECHNIQUES	7
RECEIVING SERIAL CODES – BASIC APPROACHES	8
RECEIVING SERIAL CODE – A MORE SOPHISTICATED ALGORITHM	12
TRANSMITTING SERIAL CODE	22
GENERATING PARITY	22
CONCLUSION	23

INTRODUCTION

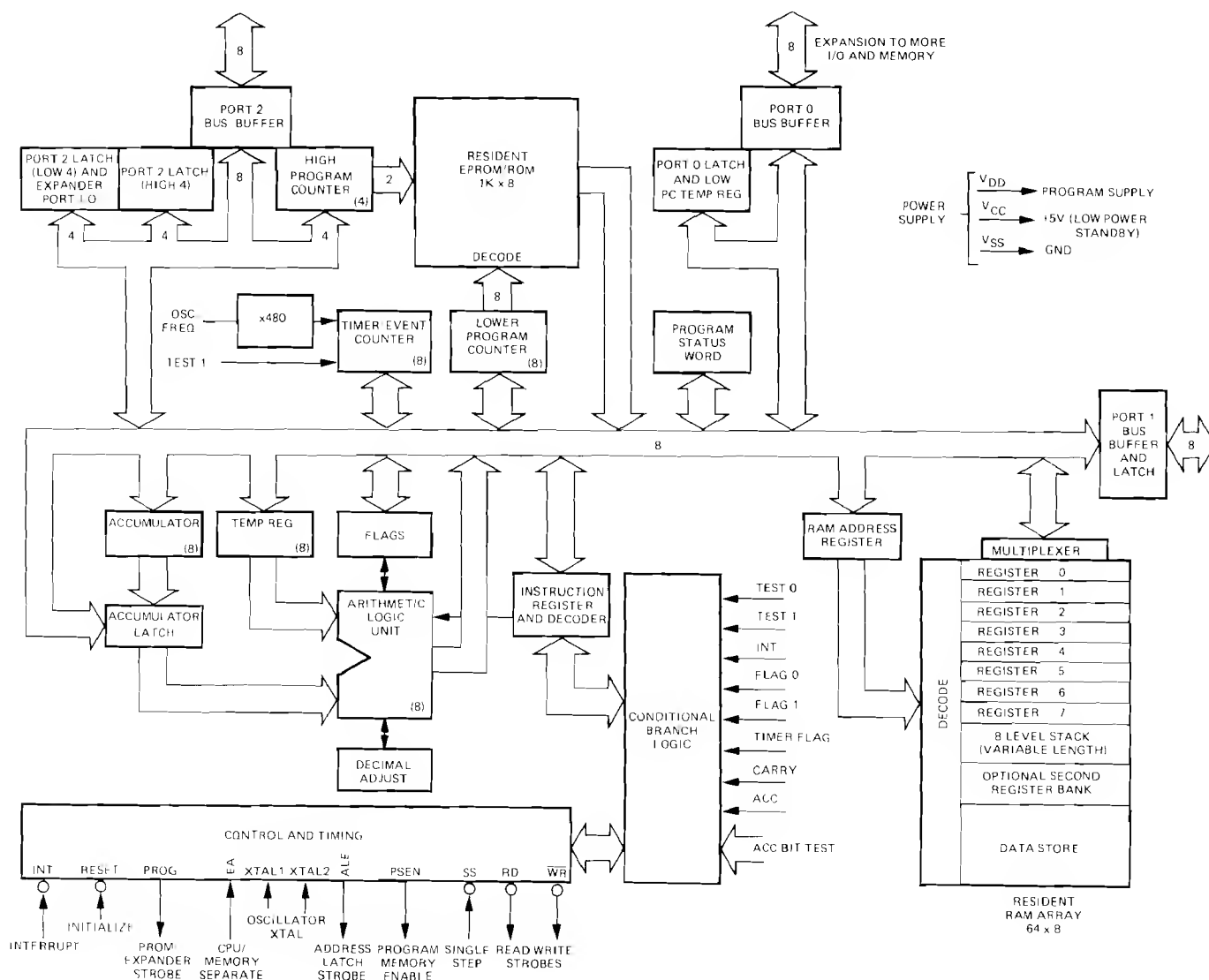
The INTEL® MCS-48™ family consists of a series of seven parts, including three processors, which take advantage of the latest advances in silicon technology to provide the system designer with an effective solution to a wide variety of design problems. The significant contribution of the MCS-48 family is that instead of consisting of integrated microcomputer components it consists of integrated microcomputer systems. A single integrated circuit contains the processor, RAM, ROM (or PROM), a timer, and I/O.

This application note suggests a variety of application techniques which are useful with the MCS-48. Rather than presenting the design of a complete system it describes the implementation of "subsystems" which are common to many micropro-

cessor based systems. The subsystems described are analog input and output, the use of tables for function evaluation, receiving serial code, transmitting serial code, and parity generation. After an overview of the MCS-48 family these areas are discussed in a more or less independent manner.

THE MCS-48™ FAMILY

The processors in the MCS-48 family all share an identical architecture. The only significant difference is the type of on board program storage which is provided. The 8748 (see Figure 1) includes 1024 bytes of erasable, programmable, ROM (EPROM), the 8048 replaces the EPROM with an equivalent amount of mask programmed ROM, and the 8035 provides the CPU function with no on board program storage. All three of these processors



MCS-48™ Internal Structure

INSTRUCTION SET

	Mnemonic	Description	Bytes	Cycle		Mnemonic	Description	Bytes	Cycles
Accumulator	ADD A, R	Add register to A	1	1	Subroutine	CALL	Jump to subroutine	2	2
	ADD A, @R	Add data memory to A	1	1		RET	Return	1	2
	ADD A, #data	Add immediate to A	2	2		RETR	Return and restore status	1	2
	ADDC A, R	Add register with carry	1	1	Flags	CLR C	Clear Carry	1	1
	ADDC A, @R	Add data memory with carry	1	1		CPL C	Complement Carry	1	1
	ADDC A, #data	Add immediate with carry	2	2		CLR F0	Clear Flag 0	1	1
	ANL A, R	And register to A	1	1		CPL F0	Complement Flag 0	1	1
	ANL A, @R	And data memory to A	1	1		CLR F1	Clear Flag 1	1	1
	ANL A, #data	And immediate to A	2	2		CPL F1	Complement Flag 1	1	1
	ORL A, R	Or register to A	1	1	Data Movers	MOV A, R	Move register to A	1	1
	ORL A, @R	Or data memory to A	1	1		MOV A, @R	Move data memory to A	1	1
	ORL A, #data	Or immediate to A	2	2		MOV A, #data	Move immediate to A	2	2
	XRL A, R	Exclusive Or register to A	1	1		MOV R, A	Move A to register	1	1
	XRL A, @R	Exclusive or data memory to A	1	1		MOV @R, A	Move A to data memory	1	1
	XRL A, #data	Exclusive or immediate to A	2	2		MOV R, #data	Move immediate to register	2	2
	INC A	Increment A	1	1		MOV @R, #data	Move immediate to data memory	2	2
	DEC A	Decrement A	1	1		MOV A, PSW	Move PSW to A	1	1
	CLR A	Clear A	1	1		MOV PSW, A	Move A to PSW	1	1
	CPL A	Complement A	1	1		XCH A, R	Exchange A and register	1	1
	DA A	Decimal Adjust A	1	1		XCH A, @R	Exchange A and data memory	1	1
	SWAP A	Swap nibbles of A	1	1		XCHD A, @R	Exchange nibble of A and register	1	1
	RL A	Rotate A left	1	1		MOVX A, @R	Move external data memory to A	1	2
	RLC A	Rotate A left through carry	1	1		MOVX @R, A	Move A to external data memory	1	2
	RR A	Rotate A right	1	1		MOVP A, @A	Move to A from current page	1	2
	RRC A	Rotate A right through carry	1	1		MOVP3 A, @A	Move to A from Page 3	1	2
Input/Output	IN A, P	Input port to A	1	2	Timer/Counter	MOV A, T	Read Timer/Counter	1	1
	OUTL P, A	Output A to port	1	2		MOV T, A	Load Timer/Counter	1	1
	ANL P, #data	And immediate to port	2	2		STRT T	Start Timer	1	1
	ORL P, #data	Or immediate to port	2	2		STRT CNT	Start Counter	1	1
	INS A, BUS	Input BUS to A	1	2		STOP TCNT	Stop Timer/Counter	1	1
	OUTL BUS, A	Output A to BUS	1	2		EN TCNTI	Enable Timer/Counter Interrupt	1	1
	ANL BUS, #data	And immediate to BUS	2	2		DIS TCNTI	Disable Timer/Counter Interrupt	1	1
	ORL BUS, #data	Or immediate to BUS	2	2	Control	EN I	Enable external interrupt	1	1
	MOVD A, P	Input Expander port to A	1	2		DIS I	Disable external interrupt	1	1
MOVD P, A	Output A to Expander port	1	2	SEL RB0		Select register bank 0	1	1	
ANLD P, A	And A to Expander port	1	2	SEL RB1		Select register bank 1	1	1	
ORLD P, A	Or A to Expander port	1	2	SEL MB0		Select memory bank 0	1	1	
Branch	INC R	Increment register	1	1		SEL MB1	Select memory bank 1	1	1
	INC @R	Increment data memory	1	1		ENTO CLK	Enable Clock output on T0	1	1
	DEC R	Decrement register	1	1	NOP	No Operation	1	1	
	JMP addr	Jump unconditional	2	2					Mnemonics copyright Intel Corporation 1976
	JMPP @A	Jump indirect	1	2					
	DJNZ R, addr	Decrement register and skip	2	2					
	JC addr	Jump on Carry = 1	2	2					
	JNC addr	Jump on Carry = 0	2	2					
	J Z addr	Jump on A Zero	2	2					
	JNZ addr	Jump on A not Zero	2	2					
	JT0 addr	Jump on T0 = 1	2	2					
	JNT0 addr	Jump on T0 = 0	2	2					
	JT1 addr	Jump on T1 = 1	2	2					
	JNT1 addr	Jump on T1 = 0	2	2					
	JF0 addr	Jump on F0 = 1	2	2					
	JF1 addr	Jump on F1 = 1	2	2					
	JTF addr	Jump on timer flag	2	2					
	JNI addr	Jump on INT = 0	2	2					
	JBb addr	Jump on Accumulator Bit	2	2					

Figure 2. 8048/8748/8035 Instruction Set

operate from a single 5-volt power supply. The 8748 requires an additional 25-volt supply only while the on board EPROM is being programmed. When installed in a system only the 5-volt supply is needed. Aside from program storage, these chips include 64 bytes of data storage (RAM), an eight bit timer which can also be used to count external events, 27 programmable I/O pins and the processor itself. The processor offers a wide range of instruction capability including many designed for bit, nibble, and byte manipulation. The instruction set is summarized in Figure 2.

Aside from the processors, the MCS-48 family includes 4 devices: one pure I/O device and 3 combination memory and I/O devices. The pure I/O device is the 8243, a device which is connected to a special 4 bit bus provided by the MCS-48 processors and which provides 16 I/O pins which can be programmatically controlled.

The combination memory and I/O devices consist of the 8355, the 8755, and the 8155. The 8355 and the 8755 both provide 2,048 bytes of program storage and two eight bit data ports. The only difference between these devices is that the 8355 contains masked program ROM and the 8755 contains EPROM. The 8155 combines 256 bytes of data storage (RAM), two eight bit data ports, a six bit control port, and a 14 bit programmable timer.

Figure 3 shows the various system configurations which can be achieved using the MCS-48 family of parts. It should also be noted that eight of the processors' I/O lines have been configured as a bidirectional bus which can be used to interface to standard Intel peripheral parts such as the 8251 USART (for serial I/O), the 8255A PPI (provides 24 I/O lines) and the complete range of memory components.

More detailed information concerning the MCS-48 family can be obtained from the "MCS-48 Microcomputer User's Manual" which provides a complete description of the MCS-48 family and its members. A general familiarity with this document will make the application techniques which follow easier to understand.

ANALOG I/O

If analog I/O is required for a MCS-48™ system there are many alternatives available from the makers of analog I/O modules. By searching through their catalogs it is possible to find almost any combination of features which is technically feasible. Perhaps the best example of such modules are the MP-10 and MP-20 hybrid modules recently introduced by Burr-Brown Research Corporation. The MP-10 provides two analog outputs and the MP-20 provides 16 analog inputs. Both of these units were

		[] Number of Available Timers () Number of Available I/O Lines			
DATA MEMORY (RAM)	1088 1K	8048 4-8155 [5] (101)	8035 8355 4-8155 [5] (116)	8048 8355 4-8155 [5] (116)	8035 2-8355 4-8155 [5] (131)
	832 768	8048 3-8155 [4] (80)	8035 8355 3-8155 [4] (95)	8048 8355 3-8155 [4] (95)	8035 2-8355 3-8155 [4] (110)
	578 512	8048 2-8155 [3] (59)	8035 8355 2-8155 [3] (74)	8048 8355 2-8155 [3] (74)	8035 2-8355 2-8155 [3] (89)
	320 256	8048 8155 [2] (38)	8035 8355 8155 [2] (53)	8048 8355 8155 [2] (53)	8035 2-8355 8155 [2] (68)
	64	8048 [1] (24)	8035 8355 [1] (28)	8048 8355 [1] (28)	8035 2-8355 [1] (43)
		1K	2K	3K	4K
		PROGRAM MEMORY (ROM)			

Figure 3. The Expanded MCS-48™ System

specifically designed to interface with microprocessors.

A block diagram of the MP-10 is shown in Figure 4. It consists of two eight bit digital to analog converters, two eight bit latches which are loaded from the data bus, and address decoding logic to determine when the latches should be loaded. The D/A converters each generate an analog output in the range of 10 volts with an output impedance of 1Ω. Accuracy is ±0.4% of full scale and the output is stable 25μsec after the eight bit binary data is loaded into the appropriate latch. The latches are loaded by the write pulse (\overline{WR}) whenever the proper address is presented to the MP-10. The lower two addresses (A_0 and A_1) are used internally by the device. Addresses A_2 & A_3 are compared with the address determination inputs B_2 and B_3 . If their signals are found to be equal, and if addresses A_4 - A_{13} are all high, then the device is selected and one of the latches will be loaded. Address bit A_1 selects between output 1 and output 2. If address bit A_0 is set then the initialization channel of the DIA is selected. In order to prepare for operation a data pattern of 80H must

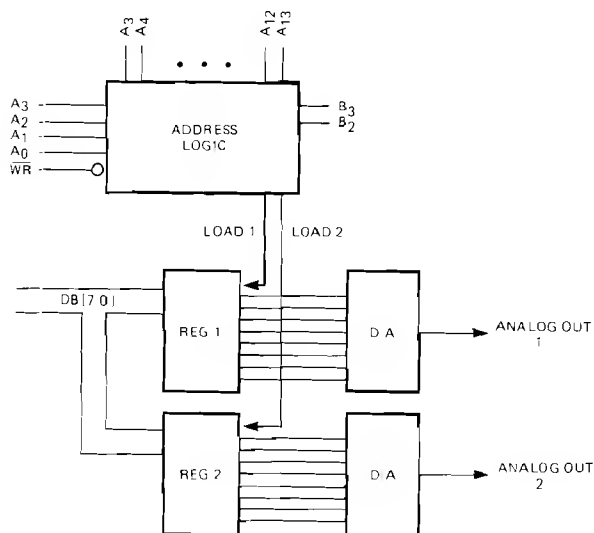


Figure 4. MP-10 Block Diagram

be output to this channel following the reset of the device.

A block diagram of the MP-20 analog to digital converter is shown in figure 5. This unit consists of a 16 input analog multiplexer, an instrumentation amplifier, an eight bit successive approximation analog to digital converter, and control logic. The 16 input multiplexer can be used to input either 16 single ended or 8 differential inputs. The output from the multiplexer is fed into the instrumentation amplifier which is configured so that it can easily be strapped for single ended 0-5 volt inputs, single ended ± 5 volt inputs, or differential 0-5 volt signals. Provisions are made for an external gain control resistor on the amplifier. The gain control equation is:

$$G = 2 + \frac{50k\Omega}{R_{ext}}$$

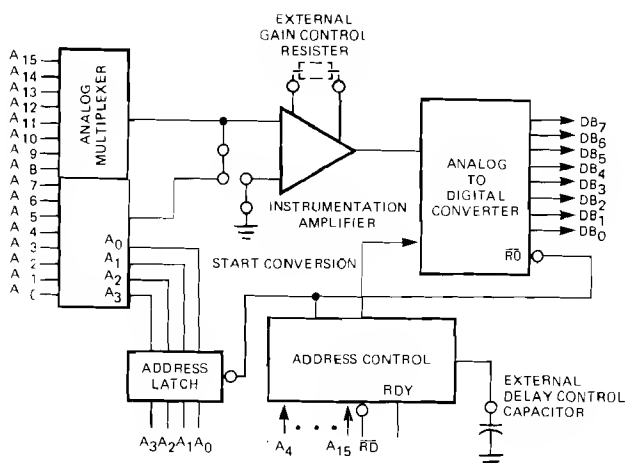


Figure 5. MP-20 Analog Subsystem

With no R_{ext} ($R_{ext} = \infty$) the gain is two and the input is 0-5 or ± 5 volts full scale. Adding an external resistor results in higher gain so that low level ($\pm 50mV$) signals from thermocouples and strain gauges can be accommodated. The output from the amplifier is applied to the actual A/D converter which provides an eight bit output with guaranteed monotonicity and an accuracy of $\pm 0.4\%$ of full scale. Note that this accuracy is specified for the entire module, not just for the converter itself. The control logic monitors address lines A15 through A4 to determine when the address of the unit has been selected. An address that the unit will respond to is determined by 11 address control pins, labeled $\overline{A4}$ through $\overline{A14}$. If one of these pins is tied to a logic 0 then the corresponding address pin must be high in order for the unit to be selected. If the pin is tied to a logic 1 then the corresponding address pin must be low. If the address of the module is selected when \overline{MEMR} pulse occurs, the lower four addresses (A3-A0) are stored in a latch which addresses the multiplexer. The coincidence of the proper address and \overline{MEMR} also initiates a conversion and gates the output of the converter on to the eight bit data bus.

The control logic of the MP-20 was designed to operate directly with an MCS-80TM system. When a \overline{MEMR} occurs and a conversion is initiated the MP-20 generates a READY signal which is used to extend the cycle of the 8080A for the duration of the conversion. READY is brought high after the conversion is complete which allows the 8080A to initiate a conversion and read the resulting data in a single, albeit long, memory or I/O cycle. The conversion time of the MP-20 depends on the gain selected for the amplifier. With no external resistor ($R = \infty$) the gain is two and the conversion time is 35 μsec . For $R = 510\Omega$ the gain is:

$$G = 2 + \frac{50k\Omega}{.51k\Omega} \cong 100$$

and the conversion time becomes 100 μsec . These settling times are specified in the MP-20 data sheet and range from 35 to 175 microseconds. The READY timing is controlled by an external capacitor. For a gain of 2 no external capacitor is required but if higher gains are selected a capacitor is needed to extend the timing.

A schematic showing both the MP-10 D/A and the MP-20 A/D connected to the 8748 is shown in Figure 6. This configuration, which consists of only four major components, gives an excellent example of what modern technology can do for

the system designer. The four components provide:

- An eight bit microprocessor
- 64 bytes of RAM
- 1024 bytes of UV erasable PROM
- A timer/event counter
- 16 digital I/O pins
- 2 testable input pins
- An interrupt capability
- 16 eight bit analog inputs
- 2 eight bit analog outputs

The MCS-48 communicates with the D/A and A/D converters in a memory mapped mode (i.e., it treats the devices as if they were external RAM). By setting an address in either R0 or R1 and then executing a MOVX the software can transfer data between the accumulator and the analog I/O. When the MCS-48 executes the MOVX instruction it first sends the eight bit address out on the bus and strobes it into the 8212 latch with the ALE (Address Latch Enable) signal. After the address is latched, the MCS-48 uses the same bus to transfer data to or from the accumulator. If data is being sent out (MOVX @Rj, A) the \overline{WR} strobe is used; if the data is being moved into the accumulator (MOVX A, @Rj) the \overline{RD} strobe is used. The one shots on the \overline{WR} line are used to delay the write strobe of the MCS-48 to meet the data set up specifications of the MP-10.

In order to provide reset capability for the analog devices without dedicating an I/O pin from the MCS-48, special addresses are used as reset channels. Executing any MOVX with an address of 0XXXXXXX will reset the A/D module; a similar operation with an address of X1XXXXXX will reset the D/A; a MOVX with an address of 01XXXXXX will reset both devices. All data transfers are accomplished with the upper two bits of the address field equal to 10. A summary of the addressing of the analog devices is shown in Table 1. Notice that except for an initialization channel for the D/A (which must

be written to following a reset to initialize its internal logic) all channels involve some form of data transfer.

As was mentioned previously, the MP-20 was designed to use the READY line of the 8080A. Obviously this presents a problem since the MCS-48 does not support a READY line (with its attendant requirement of entering WAIT state). The necessity of a READY input can be overcome by performing a read operation to set the channel address, waiting the required delay (35 μ sec for a gain of two) and then performing a second read to actually obtain the data. The second read will read in the data from the channel selected by the first read irrespective of the channel selected for the second read. Thus it is possible to use the second read to set up the channel for the third read. Each read can read in the current channel and select the next channel for conversion.

The MP-20 is shown in Figure 6 strapped to input 16 single ended ± 5 volts signals. Programs which were used to test this configuration are shown in Figure 7. The first of these programs uses the D/A converter to generate sawtooth waveforms by outputting an incrementing value to the D/A converters. The second program scans the analog inputs and stores their digital values in a table located in RAM.

Table 1. Analog Interface Addresses

INPUT OR OUTPUT		
0 X X X	X X X X	Reset A/D
X 1 X X	X X X X	Reset D/A
INPUT		
0 0 1 1	n n n n	Read A/D Channel n n n n
OUTPUT		
1 0 1 1	0 0 0 1	Initialize D/A
1 0 1 1	0 0 0 0	Write Channel 1
1 0 1 1	0 0 1 0	Write Channel 2

```

LOC  OBJ      SEQ      SOURCE STATEMENT
      0
      1
      2 ;
      3 ; TEST PROGRAM FOR ANALOG OUTPUT
      4 ; THIS PROGRAM OUTPUTS A SAW-
      5 ; TOOTH WAVEFORM BY OUTPUTTING
      6 ; AN INCREMENTING PATTERN.
      7 ;
      8
      9 ;
     10 ; EQUATES
     11 ;
     12
     13 INITCH EQU 0B3H ; D/A INITIALIZATION CHANNEL
     14 INITDT EQU 0B8H ; D/A INITIALIZATION DATA
     15 DATCH EQU 0B8H ; D/A DATA CHANNEL
     16
     17 ;
     18 ; START OF TEST
     19 ;
    0100
     20 ORG 100H ; INITIALIZE D/A
     21
    0100 2300 22 START: MOV A, #INITDT
    0102 B8B3 23 MOV RB, #INITCH
    0104 98 24 MOVX @RB, A
     25 ; TEST LOOP-OUTPUT SAWTOOTH
    0105 B8B8 26 LOOP: MOV RB, #DATCH
    0107 17 27 INC A
    0108 98 28 MOVX @RB, A
    0109 24B5 29 JMP LOOP
     30 ; END OF PROGRAM
     31 END

```

Figure 7a. D/A Exercise Program

LOC	HEX	SRV	SOURCE STATEMENT
			0
			1
			2
			3 ; TEST PROGRAM FOR ANALOG INPUT
			4 ; THIS PROGRAM SCANS THE INPUT CHANNELS
			5 ; AND STORES THE READINGS IN A TABLE
			6 ; STARTING AT BUFF.
			7 ;
			8
			9 ;
			10 ;
			11 ;
			12
0100			13 BUFF EQU 20H ; START OF BUFFER
0101			14 MAXH EQU 15 ; NO OF ANALOG INPUTS
0102			15 ADDR EQU 00H ; BASE ADDRESS OF ANALOG INPUTS
0103			16 TCLK EQU 5 ; EXECUTION TIME OF LDR
			17
			18 ;
			19 ; START OF TEST
			20 ;
			21
0100			22 ORG 100H ; SETUP TO SCAN ANALOG INPUTS
0101 B92F			23 START: MOV R3, #BUFF+MAXH
0102 B90F			24 MOV R3, #MAXH
0104 B6BF			25 MOV R0, # (ADDR+MAXCH)
			26 ; SELECT CHANNEL 15
0106 B6			27 MOVX A, @R0
			28 ; WAIT >40 MICROSECONDS
0107 B00E			29 MOV R4, #40/TICK
0109 B009			30 LDR R4, \$
			31 ; NOW SCAN ANALOGS
010B C6			32 LOOP: LDR R0
			33 ; GET DATA
010C B6			34 MOVX A, @R0
			35 ; MOVE INTO BUFFER
010D A1			36 MOV @R1, A
			37 ; DECREMENT BUFFER POINT
010E C9			38 DEX R1
			39 ; PAD 20 MICROSEC
010F B004			40 MOV R4, #20/TICK
0111 EC11			41 LDR R4, \$
			42 ; LOOP UNTIL DONE
0113 B603			43 LDR R3, LOOP
			44 ; REPEAT TEST FOREVER
0115 240C			45 JMP \$+20
			46 ; END OF PROGRAM
			47 END

Figure 7b. A/D Exercise Program

TABLE LOOKUP TECHNIQUES

In the previous section the interface between analog I/O devices and the MCS-48™ was discussed. In many applications involving analog I/O one quickly finds that nature is inherently nonlinear, and the mathematics involved in 'linearizing it' can tax the computational power of the microprocessor, particularly if it has other tasks to perform. Problems of this nature are good candidates for the use of tables.

As an example of how tables can be used as part of an analog output scheme, consider a system which requires an MCS-48 to output a variable frequency sinusoidal waveform. One method of performing this function would be to use the timer to generate an interrupt at a fixed rate of 256 times the desired output frequency. At each interrupt the appropriate value of the sine function could be calculated from the MacLaurin series:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \cdots \frac{(-1)^k x^{2k+1}}{(2K+1)!}$$

Where K is chosen to be large enough to provide the required accuracy.

The above calculation, although conceptually simple, would be time consuming and would severely limit the possible output frequencies which could be obtained. As an alternative to calculating these values in real time, the values could be precalculated off line and stored in a table. Upon each interrupt the MCS-48 would merely have to retrieve the appropriate value from the table and output it to the D/A converter. The MCS-48 provides a special instruction which can be used to access data in a table. If the table is stored in the last 256 bytes of the first kilobyte of MCS-48 memory then the table lookup can be performed by loading the independent variable (time in this case) into the accumulator and executing the instruction.

MOVP3 A, @ A

This instruction uses the initial contents of the accumulator to index into page 3 of program storage. The location pointed to is read and the contents placed in the accumulator. If (as is often the case) a table of fewer than 256 entries is required, then the table can be located in any page of program memory and the instruction:

MOVP A, @ A

can be used to retrieve data from the table. This instruction operates in the same manner as does the previous instruction except that the current page of program storage is assumed to contain the table.

If it is possible to devote slightly more of the microprocessor's time to the table look up process, then a much smaller table can often be utilized by taking advantage of interpolation to determine values of the function between values which are actual entries in the table. As an example of this

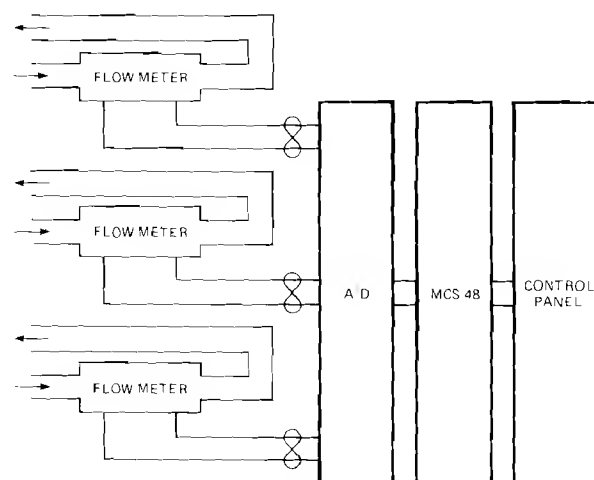


Figure 8. Flow Monitoring System

process consider the hypothetical system shown in Figure 8. The purpose of this system is to measure the flow through the three pipes, add them, and display the total flow on the control panel. The system consists of three flow meters which generate a differential voltage which is some function of flow, an A/D system with at least three differential inputs, an MCS-48, and a control panel. The schematic shown in Figure 6 could easily become part of this system, with the spare digital I/O of the MCS-48 used as an interface to the control panel. The simplicity of this system is clouded by the flow transducers, which are assumed to be not only nonlinear but also to require individual calibration (this is not an unreasonable assumption for a flow transducer). By using a table look up process and an 8748 the flow transducers can be calibrated and the results of the calibration tests stored directly in tables in the 8748. (The 8748 has a PROM in place of the ROM of the 8048 and thus makes such 'one off' programming practical.)

The results which might be obtained from calibrating one of the flow meters is shown in Figure 9. The results are plotted as gals/hour versus the measured voltage generated by the transducer. The voltage is shown in hexadecimal form so that it corresponds directly to the digital output of the analog to digital converter. The flow required to generate seventeen evenly spaced voltages (00H-100H in steps of 10H) has been measured and plotted. This information is shown in tabular form in Figure 10. It is necessary to generate a program which will convert any measured input from 00H to FFH into the flow in units which can be interpreted by a human operator. This can easily be done by simple interpolation.

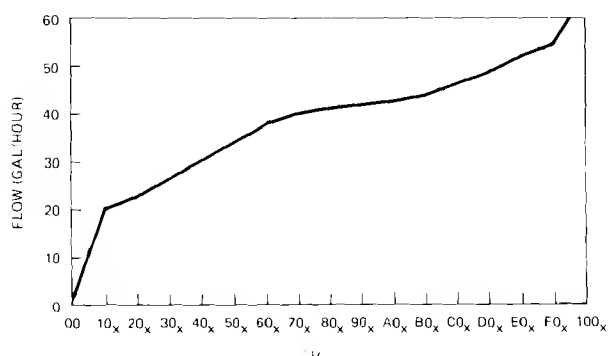


Figure 9. Flow Calibration Curve

TRANSUCER VOLTAGE (HEX)	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0	100
MEASURED FLOW (GAL/HOUR)	0	10	22	26	30	34	38	40	41	42	43	45	48	49	53	56	63

Figure 10. Tabulated Flow Data

The eight bits of independent variable (voltage) can be looked on as two four bit fields. The most significant four bits (7-4) will be used to retrieve one of the table values. The lower four bits (3-0) will be used to interpolate between this value and the value retrieved from the next higher location in the table. If the upper four bits are given the symbol I and the lower four bits the symbol N, then the interpolation can be expressed as:

$$F(x) = F(I) + \frac{N}{16} [F(I+1) - F(I)]$$

Where x is the measured voltage and F(x) is the corresponding flow.

If, as an example, the transducer voltage was measured as 48H then the flow (ref. Figure 10) would be:

$$F = 30 + \frac{8}{16} (34 - 30) = 32$$

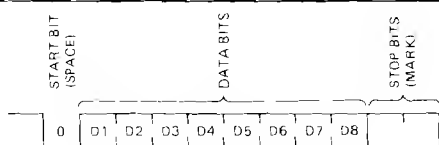
A subroutine which implements this calculation is shown in Figure 11. Before it is called the independent variable (V) is placed in the accumulator and register R1 is set to point at the first value in the table. Aside from simple additions and subtractions the only arithmetic required is to multiply two values and then divide them by 16. The multiplication is handled via a subroutine which is also shown in Figure 11. The division by 16 can be performed by a four place right shift followed by a rounding operation. The routine shown will handle a monotonic increasing function of a single independent variable. Fairly simple modifications are required for nonmonotonic functions. Functions of two variables can be handled by interpolating on a plane rather than along a straight line. Although this is more time consuming, requiring an interpolation for each of the independent variables and a third to interpolate the final answer, it still provides a simple means of quickly calculating the required function. The use of tables can offer a powerful technique for function evaluation to the designer.

RECEIVING SERIAL CODE—BASIC APPROACHES

Many microprocessor based systems require some form of serial communication. Serial communication is extensively used because it allows two or more pieces of equipment to exchange information with a minimal number of interconnecting wires. The minimization of interconnecting wires results in simpler, cheaper, interconnects because fewer (or smaller) cables and connectors are required. Since the required number of drivers and receivers required is reduced, it can become economically feasible to provide much higher noise immunity

LOC	OBJ	SOURCE STATEMENT	LOC	OBJ	SOURCE STATEMENT
0		*****	56	REL	
1			57		
2		APPROX	58		
3		AT ENTRY R1 POINTSAT TABLE	59	-----	
4		A HAS INDEPENDANT VARIABLE	60	MULTIPLY	
5			61	-----	
6		*****	62		; SET UP COUNT AND AEX
7			63	MULT: MOV	COUNT, #8
8		-----	64	MOV	AEX, #0
9		SWAP	65		; CLEAR CARRY
10		-----	66	LOOPA: CLR	C
11			67		; IF MULTIPLIER[0] <> 1 THEN SHIFT PRODUCT
0000		12 RX0 EQU R0 ; POINTER 0	68	LOOPB: JBO	SSUM
0001		13 RX1 EQU R1 ; POINTER 1	69	XCH	A, AEX
0002		14 AEX EQU R2 ; EXTENSION OF A REGISTER	70	RJC	A
0003		15 COUNT EQU R3 ; COUNTER	71	XCH	A, AEX
0004		16 TEMP EQU R4 ; TEMP STORAGE	72	RJC	A
		17	73		; LOOP UNTIL DONE
		18 ; -----	74	DJNZ	COUNT, LOOPB
		19 ; APPROXIMATION	75	RLT	
		20 ; -----	76		; ELSE ADD MULTIPLIER AND SHIFT PRODUCT
		21	77	SEGN: XCH	A, AEX
0100		22 ORG 100H	78	ADD	A, #RX0
		23 ; POINT RX0 AT TEMP	79	RJC	A
0100 5004		24 APPROX: MOV RX0, #TEMP	80	XCH	A, AEX
		25 ; TEMP=N AND 0FH	81	RJC	A
		26 ; A=P AND 0FH	82		; LOOP UNTIL DONE
0102 0000		27 MOV @RX0, #0	83	DJNZ	COUNT, LOOPA
0104 30		28 XCHD A, @RX0	84	RLT	
0105 47		29 SWAP A	85		
		30 ; RX1=BASE+A	86		
0106 69		31 ADD A, RX1	87	-----	
0107 A9		32 MOV RX1, A	88	TABLE TO TEST PROGRAM	
		33 ; RX1=TABLE(P)	89	-----	
		34 ; A=TABLE(P+1)	90		
0108 E3		35 MOVFP A, #A	91	ORG 380H	
0109 29		36 XCH A, RX1	92		
010A 17		37 INC A	0380 00	93	TABLE: DB 00 ; THIS TABLE IS FROM FIG 10
010B E3		38 MOVFP A, #A	0381 0A	94	DB 10
		39 ; A=TABLE(P+1)-TABLE(P)	0382 16	95	DB 22
010C 37		40 CPL A	0383 1A	96	DB 26
010D 69		41 ADD A, RX1	0384 1E	97	DB 30
010E 37		42 CPL A	0385 22	98	DB 34
		43 ; A=n*A/16	0386 26	99	DB 38
010F 341D		44 CALL MULT	0387 28	100	DB 40
0111 5002		45 MOV RX0, #AEX	0388 29	101	DB 41
0113 30		46 XCHD A, @RX0	0389 2A	102	DB 42
0114 47		47 SWAP A	038A 2B	103	DB 43
0115 2A		48 XCH A, AEX	038B 2D	104	DB 45
0116 7219		49 JBO ADJUST	038C 30	105	DB 48
0118 2A		50 XCH A, AEX	038D 31	106	DB 49
0119 2A		51 ADJUST: XCH A, AEX	038E 35	107	DB 53
011A 17		52 INC A	038F 38	108	DB 56
		53 ; A=n*TABLE(P)	0390 3F	109	DB 63
011B 69		54 ADD A, RX1	110		
		55 ; RETURN	111	END	

with more sophisticated (and expensive) line terminators. The final, and usually most persuasive, argument in favor of serial communication is that it may be the only method available to accomplish the job. The obvious example of this is telecommunications where it is necessary to encode parallel information into serial format in order to communicate via the telephone network. The intent of this section is to show how the facilities of the MCS-48TM can be brought to bear on the problem of serial communication.



Probably the most common form of serial communication is that used by the ubiquitous Teletype-serial ASCII. This format, shown in Figure 12, consists of a START bit (0 or SPACE) followed by eight data bits which are in turn followed by two STOP bits (1 or MARK). In actual practice the

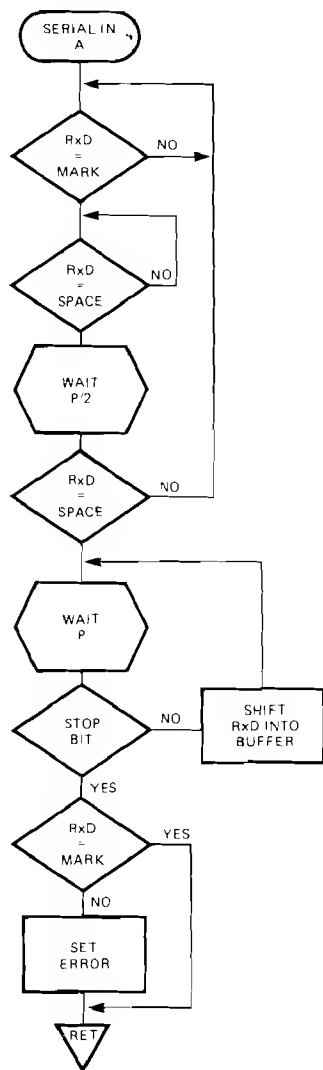


Figure 13. Sample Serial Input Routine

invalid and the process is reinitialized. If the line is still a SPACE, then the START bit is assumed to be valid and a delay of one bit time is started. At the completion of the delay the first data bit is sampled and a new delay of one bit time is initiated. This process is repeated until all eight data bits have been sampled. The last bit sampled is checked to determine if it is a valid STOP bit (a MARK). If it is, the character is assumed to be valid; if it is not, the character has a framing error and is probably invalid. A listing of a program which implements the above procedure is shown in Figure 14.

A disadvantage of the approach outlined in Figure 13 is that while the processor is inputting data serially it must totally dedicate itself to this task. Accurate timing can only be maintained if the program remains in a tight wait loop without allowing itself to be diverted to other functions. During reception of a character from a Teletype

the processor will spend only a 100μsecs or so processing data and the rest of the 100 millisecs waiting to do the processing at the right time. This lack of efficiency (approximately 0.1%) in the utilization of processing power is why devices such as the 8251 USART find broad application in micro-processor systems.

LOC	OBJ	SEQ	SOURCE STATEMENT
0			*****
1			;
2			SIMPLE SERIAL INPUT
3			-THIS CODE ASSUMES RxD IS
4			CONNECTED TO PIN T8
5			;
6			*****
7			;
8			-----
9			EQUATES
10			-----
11			;
0002		12	COUNT EQU R2 ; COUNTER
0008		13	BITNO EQU 8 ; NO OF BITS TO RECEIVE
0002		14	DLVHI EQU 2 ; HI DLY COUNT
00A4		15	DLYLO EQU 0A4H ; LO DLY COUNT
16			;
0100		17	ORG 100H
18			;
0100 2600		19	SERIN: JNT0 \$; LOOP UNTIL RxD=MARK
20			;
0102 3602		21	J10 \$; NOW LOOP UNTIL RxD=SPACE
22			;
0104 341C		23	CALL HBIT ; WAIT 1/2 BIT TIME
24			;
0106 3600		25	J10 SERIN ; IF FALSE START REINITIALIZE
26			;
0108 BA09		27	MOV COUNT, BITNO-1 ; ELSE SET BIT COUNT
28			;
010A 341C		29	LOOP: CALL HBIT ; WAIT 1 BIT TIME
010C 341C		30	CALL HBIT
31			;
32			;
33			;
010E FA15		34	DJNZ COUNT, LOAD ; DECREMENT COUNT
0110 97		35	CLR C ; -IF ZERO EXIT WITH CARRY SET ON
0111 3614		36	JT0 EXIT ; -FRAMING ERROR
0113 A7		37	CPL C
0114 83		38	EXIT: RET
39			;
0115 97		40	LOAD: CLR C ; LOAD DATA
0116 2619		41	JNT0 LLLA
0118 A7		42	CPL C
0119 67		43	LLLA: RSC A ; AND LOOP
44			;
011A 240A		45	JMP LOOP
46			;
47			-----
48			DELAY ONE HALF BIT TIME
49			-----
50			;
51			;
011C BC02		52	HBIT: MOV R4, DLVHI ; SET UP LOOP
53			;
011E BBA4		54	HLOOP: MOV R3, DLYLO ; LOOP UNTIL TIME DONE
0120 EB20		55	DJNZ R3, \$
0122 EC1E		56	DJNZ R4, HLOOP
0124 83		57	RET
58			;
59			END ; END OF PROGRAM

Figure 14. Simple Serial Input

The 8251 USART is simple to interface to the MSC-48. Figure 15 shows such an interface. The USART requires a high speed clock (CLK), an initialization signal (RESET), data clocks (TxC and RxC), and data in order to operate. A circuit showing the connection of an 8748 to an 8251 USART is shown in Figure 15. In the circuit shown the high speed clock (which is used for internal sequencing by the USART) is provided by con-

LOC	SEQ	SOURCE STATEMENT
	0	;
	1	;
	2	;
	3	;
	4	;
	5	;
	6	;
	7	;
	8	;
	9	;
	10	;
0020	11	MCLR EQU 20H ; USART RESET ADDRESS
0001	12	DLY EQU 01h ; USART RESET DELAY
0071	13	UCON EQU 7FH ; USART CONTROL ADDRESS
00C1	14	MODE EQU 0CEH ; USART MODE
0021	15	CMD EQU 21h ; USART CMD
0071	16	STAT EQU 7FH ; USART STATUS
0001	17	VAL EQU R1 ; TEST VALUE
00B1	18	MASK EQU 0BFH ; CHANGES CMD TO DATA CHANNEL
	19	;
0100	20	ORG 100H
	21	;
	22	;
	23	TEST: ENT0 CLK
0101 75	24	ORL P2,#MCLR
0103 BAE1	25	MOV R2,#DLY
0105 EAB5	26	LOOP: DJNZ R2,LOOP
0107 9ADF	27	ANL P2,#(NOT MCLR)
	28	;
	29	;
0109 237E	30	MOV A,#UCON
010B 3A	31	OUTL P2,A
	32	;
	33	;
010C 23C6	34	MOV A,#MODE
010E 90	35	MOVX @R0,A ; (CONTENTS OF R0 UNIMPORTANT)
010F 2321	36	MOV A,#CMD
0111 90	37	MOVX @R0,A
	38	;
	39	;
	40	;
	41	;
	42	;
	43	;
	44	TLP: MOV A,#STAT
0112 237E	45	OUTL P2,A
0114 3A	46	MOVX A,@R0 ; (CONTENTS OF R0 UNIMPORTANT)
0115 00	47	RRC A
0116 67	48	JNC TLP
0117 E612	49	MOV A,VAL
0119 F9	50	ANL P2,#MASK
011A 9ADF	51	MOVX @R0,A
011C 90	52	INR VAL
011D 19	53	JMP TLP
011E 2412	54	;
	55	END

Figure 16. 8251 Test Program

R0 or R1. In order to minimize the circuitry in Figure 15 an approach utilizing some of the I/O pins of the MCS-48 to address the 8251 was chosen instead. By connecting the chip select (\overline{CS}) input of the 8251 to bit 7 of port 2 (P27) and similarly connecting the C/\overline{D} address line of the 8251 to bit 6 of port 2 (P26) it is possible to address the 8251 without using R0 or R1. The instruction sequence to access the 8251 is to first reset P27 and set P26 to the appropriate state, use a MOVX instruction to perform the appropriate operation, and then finally set P27 to deselect the 8251. As a concrete example of this addressing, Figure 16 shows the code necessary to initialize the 8251 and output an incrementing test pattern on a status driven basis.

If more than one 8251 were to be added to the MCS-48, or if other types of peripheral circuitry would be required (e.g. an 8253 timer to generate the data clocks) it would probably become desirable

to add the circuitry necessary to use R0 or R1 to address the peripheral devices. The circuitry which has to be added to Figure 15 in order to make use of R0 or R1 to address the USART is shown in Figure 17. Note that only the changes to Figure 15 are shown. The additional component required is the 8212 eight bit latch. This latch is loaded, whenever a valid address is on the bus by the Address Latch Enable (ALE) signal provided by the MCS-48. During an external read or write cycle this address is used to address the 8251 in a linear select mode. In the circuit shown, the 8251 will be selected by any address with bit 1 a logical zero (XXXXXX0X) and the selection of control or data transfer (C/\overline{D}) will be based on bit zero of the address obtained from R0 or R1. Figure 18 shows the program of Figure 16 modified to utilize the addressing inherent in the MOVX instructions.

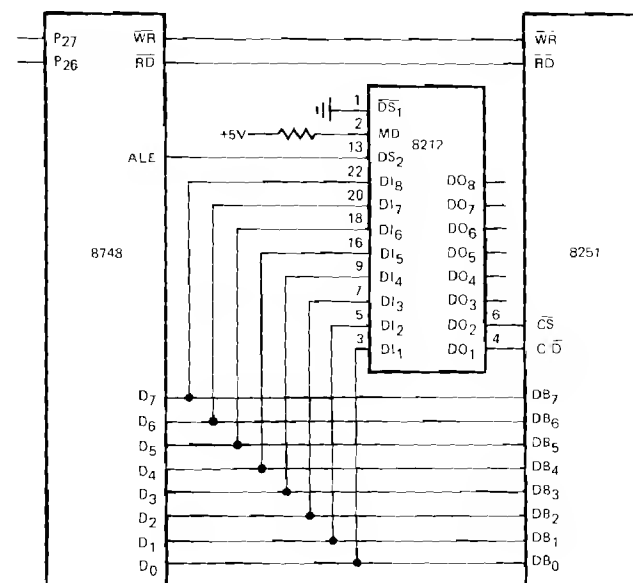


Figure 17. Modified MCS-48 to 8251 Interface

RECEIVING SERIAL CODE—A MORE SOPHISTICATED ALGORITHM

Although the USART does an admirable job of performing the serial I/O function for the MCS-48™, there are some situations where it can not be used. These situations may be caused by economic factors, such as an extremely cost sensitive design, or because the code which must be utilized cannot be accommodated by the USART. An example of of such a code will be discussed later. Recall that the principal objection to the approach to serial input shown in Figure 13 was that it consumes much of the processor's power by merely spinning in loops in order to wait preset time delays.

LOC	OBJ	SEQ	SOURCE STATEMENT
		0	;
		1	;
		2	;
		3	;
		4	;
		5	;
		6	;
		7	;
		8	;
		9	;
		10	;
0020		11	MCLR EQU 20H ; USART RESET ADDRESS
0001		12	DLY EQU 01H ; USART RESET DELAY
0003		13	UCON EQU 03H ; USART CONTROL ADDRESS
000E		14	MODE EQU 0CEH ; USART MODE
0021		15	CMD EQU 21H ; USART CMD
0003		16	STAT EQU 03H ; USART STATUS
0001		17	VAL EQU R1 ; TEST VALUE
0000		18	DATA EQU 00 ; USART DATA ADDRESS
		19	;
0100		20	ORG 100H
		21	;
		22	;
0100 75		23	TEST: EXT0 CLK ; TURN ON CLOCK
0101 0A20		24	ORL P2,#MCLR ; AND RESET USART
0103 0A01		25	MOV R2,#DLY
0105 0A05		26	LOOP: DJNZ R2,LOOP
0107 9ADF		27	ANL P2,#(NOT MCLR)
		28	;
0109 2303		29	MOV A,#UCON ; SELECT USART CONTROL
		30	;
010B 23CE		31	MOV A,#MODE ; SEND MODE AND COMMAND
010C 90		32	MOVX @R0,A ; (CONTENTS OF R0 UNIMPORTANT)
010E 2321		33	MOV A,#CMD
0110 90		34	MOVX @R0,A
		35	;
		36	;
		37	;
		38	;
		39	;
		40	;
		41	;
		42	;
0111 2303		43	TLP: MOV A,#STAT
0113 90		44	MOVX A,@R0 ; (CONTENTS OF R0 UNIMPORTANT)
0114 67		45	RRC A
0115 5611		46	JNC TLP
0117 F9		47	MOV A,VAL
0118 0800		48	MOV R0,#DATA
011A 90		49	MOVX @R0,A
011B 19		50	INC VAL
011C 2411		51	JMP TLP
		52	;
		53	END ; END OF PROGRAM

Figure 18. Modified 8251 Test Program

The timer resident on the MCS-48 provides a solution to this problem. Instead of spinning in a loop the program can set the timer for a given interval, start it, and proceed to other tasks. When the timer overflows, an interrupt will be generated to notify the software that the present time period has elapsed. An extension of the algorithm of Figure 13 which uses the timer in this fashion is shown in Figure 19. This algorithm is identical to the preceding one up until the detection of the leading edge of the start bit. At this point the timer is set to one half of the bit time (P) and a return is made to the calling program which can start additional processing. At the completion of this time interval a timer overflow interrupt is generated. When the first interrupt is detected, the serial line is checked to ensure that it is in a spacing condition (valid START bit). If it is, the timer is set to P (to sample the middle of the first data bit) and a return is made to the program which was running when the

interrupt occurred. If the serial line has returned to the MARK state, a status flag is set to indicate an error and a return is made. On subsequent interrupt detection, the data is sampled, the timer is reinitiated, and control is returned to the program which was running when the interrupt occurred. When the last (i.e. STOP) bit is detected a completion flag is set and a return is made to the program running when the timer overflow occurred. By periodically checking the error and completion flags the running program can determine when the interrupt driven receive program has a character assembled for it.

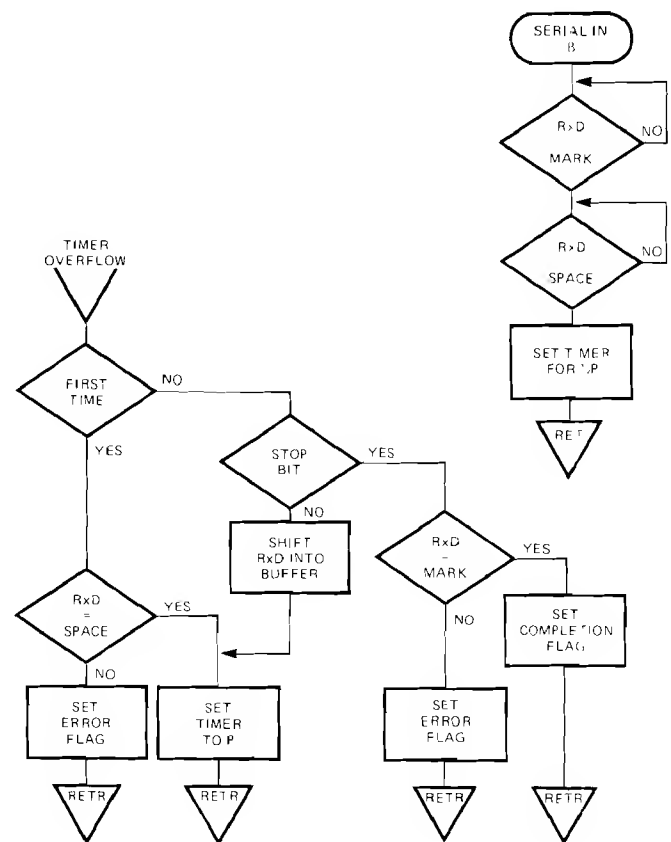


Figure 19. Improved Serial Input Routine

Using the timer to implement time delays as shown in Figure 19 results in considerable savings in processing time; two problems remain, however, which must be solved before an adequate software solution to the problem of receiving serial code can be found. The first problem is that even though the delays between bit samples are implemented via the timer rather than program loops the loop construction is still used to detect the leading edge of

the START bit. Although this results in the waste of processing power, the second problem is even more serious. For longer messages the required accuracy of the clocks becomes more and more stringent. Using the sampling technique discussed a cumulative error of one half a bit time in the time at which a bit sample is taken will result in erroneous reception. The maximum timing error which can be tolerated and yet still allow proper detection of an 11 bit ASCII character is then:

$$E_{\max} = \frac{0.5 \cdot \text{BIT TIME}}{\text{CHARACTER TIME}} - \frac{0.5P}{11P} = 4.5\%$$

where P is the period of single bit. The corresponding calculation for a 32 bit character yields:

$$E_{\max} = \frac{0.5P}{32P} = 1.6\%$$

Since this calculation does not allow for distortion on the signals, it is obvious that either extremely stable clocks will be required or a more tolerant algorithm must be devised. This problem is particularly serious at relatively high baud rates where the resolution of the counter (80μsecs with a 6 MHz crystal) becomes a significant percentage of the period of the received signal. At the 110 baud rate of the Teletype the 80μsec resolution of the clock allows a maximum accuracy of 0.33%; at 2400 baud this figure is reduced to 3.8%.

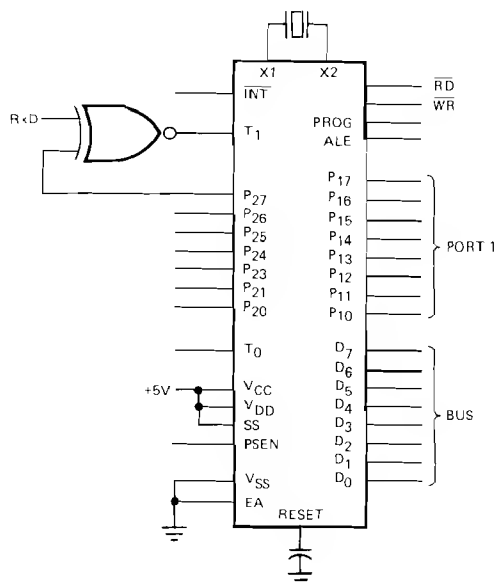


Figure 20. Detecting RxD Edges

Both efficient detection of the start bit and increased timing accuracy can be obtained if the MCS-48 can detect edges on the incoming received data (RxD). A hardware construct which allows this is shown in Figure 20.

The received data (RxD) is Exclusive NORed with bit seven of port two and fed into the TEST (T1) pin of the MCS-48. By manipulating P27 the program can now cause T1 to be either RxD or $\overline{\text{RxD}}$. (If P27 = 1 then T1 = RxD; if P27 = 0 then T1 = $\overline{\text{RxD}}$.) Note that not only can T1 be tested directly by the software but that it is the input which is used when the MCS-48 timer is in the event counter mode. The significance of this will be discussed later. The relationship between T1, P27, and RxD is given by the Boolean expression:

$$\overline{\text{T1}} = \text{P27} \cdot \overline{\text{RxD}} + \overline{\text{P27}} \cdot \text{RxD}$$

Figure 21 flowcharts a means of utilizing this hardware construct to avoid the necessity of wasting time in program loops to detect the leading edge of the start bit. The receive operation is initialized when the program desiring to receive serial data calls the INIT subroutine (Figure 21a). Since INIT is going to manipulate the timer the first action it performs is to disable the timer overflow interrupt. Its next step is to set P27 to a logical 1. Setting P27 in this manner causes the TEST 1 input to the MCS-48 to follow $\overline{\text{RxD}}$. By setting up the receive circuitry in this manner a high to low transition will occur on TEST 1 when the RxD goes from the MARKING to SPACING state (i.e. the START

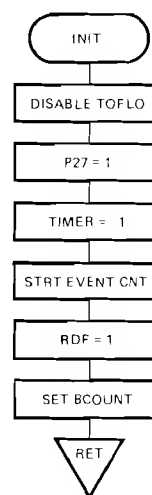


Figure 21a. Interrupt Driven Serial Receive Flowchart

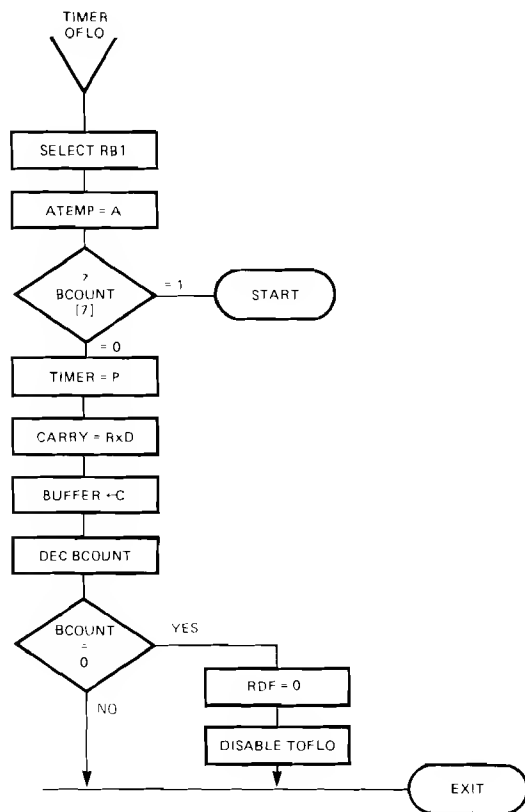


Figure 21b. Interrupt Driven Serial Receive Flowchart

bit occurs). By setting the timer to 0FFH and enabling it in the event count mode, the INIT routine sets up the MCS-48 to generate a timer overflow interrupt on the next MARK to SPACE transition of RxD (the TEST 1 input doubles as the event counter input). Before returning to the calling program the INIT routine sets a flag (RDF) which will be cleared by the receive program when the requested receive operation is complete. INIT also sets a value into a register called BCOUNT. The receive program interprets BCOUNT as follows:

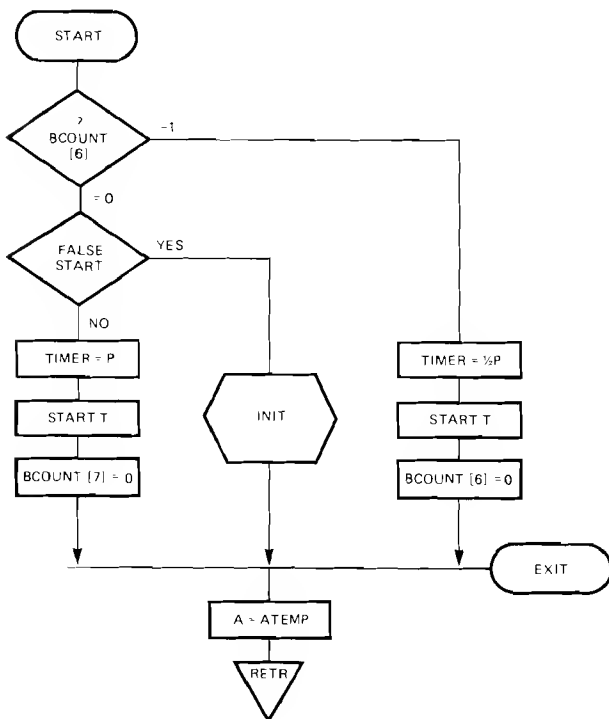
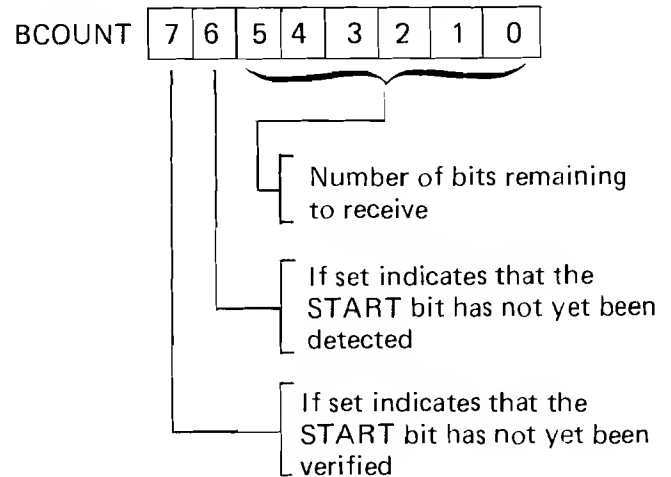


Figure 21c. Interrupt Driven Serial Receive Flowchart

In order to request the reception of the 11 bit ASCII code INIT would set BCOUNT to 11001011B. The start bit has been neither verified nor detected and 11 bits (1011B) are required.

After INIT is called the reception of the individual serial data bits will proceed on an interrupt driven basis until a complete character has been assembled. When this occurs the interrupt driven program will set the RDF (Receive Done Flag) to a zero to indicate that it has completed the requested operation and then terminate itself. The procedure which is used to accomplish this is shown in Figures 21b and 21c.

Since all operations of this program are the result of the occurrence of a timer overflow interrupt, it is necessary to briefly review the interrupt structure of the MCS-48. There are two sources of interrupt; an external interrupt which is the result of a logical zero signal applied to the INT pin of the MCS-48, and an internal interrupt which is caused by a timer overflow condition. The timer overflow occurs whenever the timer is incremented from 0FFH to zero whether it be in the timer or event count mode. When one of these events occurs the hardware in the MCS-48 forces the execution of a CALL. This CALL has a preset address of location 3 if it is due to the external interrupt and location 7 if it is due to a timer overflow. If both of these

events occur simultaneously the external interrupt will take precedence. The CALL automatically saves the contents of the program counter for the running program and its PSW (program status word) on a stack the hardware maintains in RAM locations 8-23. Although the hardware saves the program counter and PSW, it remains the responsibility of any interrupt driven software to make absolutely certain that it does not modify any memory locations or registers which are being used by the main program. The most convenient way of ensuring this in the MCS-48 is to dedicate the second bank of registers (RB1) to the interrupt driven program. One of these registers has to be used to save the accumulator (which is not part of the register bank) but seven registers remain; including two which can be used as pointers to the rest of the RAM (R0 and R1). Note that if this approach is taken then these registers have to be allocated between the program which services the external interrupt and the one which services the timer overflow. This problem is somewhat alleviated by a hardware lockout which prevents the timer overflow interrupt from interrupting the external interrupt service routine and vice versa. This is implemented by locking out new interrupts between the time an interrupt is recognized and the time a RETR instruction is executed. The RETR instruction is like a normal RET (return from subroutine) except that the PSW as well as the program counter is restored. The RETR instruction can be very much thought of as a return from interrupt instruction in the MCS-48.

The receive program under discussion uses register bank 1 in the manner described. Whenever a timer overflow occurs (e.g. on the next MARK to SPACE transition of RxD after INIT is called), control is passed (by the hardware generated CALL) to the point labeled TIMER OFLO in Figure 21b. This program segment immediately selects register bank 1 (RB1) and then saves the accumulator (A) in a location called ATEMP which is actually R7 of RB1. The program then tests bit seven of BCOUNT (R6 of RB1) to find out if a START bit has been verified (i.e. the edge of the START bit has first been detected and then verified to still be a SPACE one-half a bit time later. If BCOUNT [7] is a zero the START has been verified and the program proceeds to set the timer to P (the period of the serial bit), get the current serial data into the carry bit, and then shift the carry bit into a buffer. After saving the data the program decrements BCOUNT and tests it for zero. If BCOUNT is zero the receive operation is complete so the program sets RDF to a zero and disables timer overflow interrupts. Whether or not BCOUNT is zero, control is passed to EXIT where A is loaded with ATEMP and a

RETR is executed. Note that since the state of the flip flop which selects RB1 is saved as part of the PSW, the execution of RETR automatically selects the register bank which was active when the interrupt occurred.

If BCOUNT [7] is still set when it is tested, control is passed to START (Figure 21c) where bit 6 is tested to determine if the START has been detected yet. If BCOUNT [6] is set it indicates that this is the first occurrence of a timer overflow since the receive process was initialized by the INIT subroutine. If this is so, the program assumes that the START bit has just started and therefore it sets the timer to one-half of a bit time ($1/2 P$), starts the timer in the timer mode, and clears BCOUNT [6] to indicate that the START bit has been detected. The next overflow will again result in the execution of the program in Figure 21b and again BCOUNT [7] will be found to be set. This time, however, BCOUNT [6] will be reset and the program will know that it should test the START bit to ensure that it is still a SPACE. This test is performed and if successful the timer is set for a bit period P and BCOUNT [7] is reset so that on the next occurrence of a timer overflow the program will know that it should start assembling serial bits into a character. If the test is unsuccessful, the subroutine INIT is used to reinitialize the receive program. In either case control is passed to EXIT where a return from interrupt mode occurs.

This receive program, listings of which appear in Figure 22, allows the reception of serial characters transparently to the main running software. After INIT is called the main program has only to check RDF periodically to find out if there is data in the buffer for it. It would be fairly easy to 'double buffer' this operation by providing a buffer which the receive program uses to deserialize the incoming code and a second buffer to store the assembled character. If the program would reinitialize itself upon completion, the reception of a string of characters could proceed in much the same way as it would if a status driven USART were being used.

Although this program solves the first problem of software controlled reception (lack of efficiency) the second problem—sensitivity to frequency variations—remains. An example of a code which would be susceptible to this problem is the 31,26 BCH code commonly used in supervisory control systems. (A supervisory control system is, in essence, a remote control system which allows a human or computer operator the control of a system via a serial communications link.) The BCH codes are used because of their error detection capabilities and are a class of cyclical redundancy

LOC	OBJ	SEQ	SOURCE STATEMENT	LOC	OBJ	SEQ	SOURCE STATEMENT
		0		0023 FE		71	START: MOV A,BCOUNT
		1	*****	0024 D237		72	JB6 SLLC ; DO;
		2				73	
		3	SERIAL INPUT USING THE MCS-48			74	
		4	THIS CODE ASSUMES HARDWARE	0026 5635		75	JT1 SLLD ; IF TEST1=0 THEN
		5	SHOWN IN FIG 20. TO USE			76	
		6	THIS ROUTINE CALL INIT.			77	
		7	WHEN R0F=0 THE ASSEMBLED			78	
		8	CHARACTER WILL BE IN SERBUF			79	
		9				80	
		10	*****			81	
		11				82	
		12	-----	0028 23D7		83	MOV A,#-P
		13	EQUATES	002A 62		84	MOV T,A
		14	-----	002B 55		85	STRT T
		15		002C 9A7F		86	ANL P2,#7FH
0007		16	ATEMP EQU R7 ; STORAGE FOR A DURING INTERRUPT	002E 05		87	I
0006		17	BCOUNT EQU R6 ; CONTAINS NUMBER OF BITS IN MSG	002F FE		88	MOV A,BCOUNT
0002		18	COUNT EQU R2 ; UTILITY COUNTER	0030 537F		89	ANL A,#7FH
0000		19	RX0 EQU R0 ; POINTER	0032 AE		90	MOV BCOUNT,A
0008		20	BITNO EQU 8 ; NUMBER OF BITS	0033 043F		91	JMP SEXIT
0029		21	P EQU 41 ; SAMPLE PERIOD			92	
0020		22	SERBUF EQU 20H ; SERIAL BUFFER			93	
0024		23	R0F EQU 24H ; RECEIVE DONE FLAG			94	
		24		0035 1441		95	
		25	-----			96	SLLD: CALL INIT
		26	CONTROL PASSED HERE WHEN TIMER OFLO OCCURS			97	
		27	-----			98	
		28				99	
0007		29	ORG 07H			100	
		30				101	
0007 D5		31	IMVEC: SEL R01 ; /*ENTER INTERRUPT MODE*/	0037 23BC		102	
0008 AF		32	MOV ATEMP,A	0039 62		103	SLLC: MOV A,#-(P/2)
		33		003A 55		104	MOV T,A
0009 FE		34	MOV A,BCOUNT ; IF BCOUNT(7)=0 THEN	003B FE		105	STRT T
000A F223		35	JB7 START	003C 53BF		106	MOV A,BCOUNT
		36		003E AE		107	ANL A,#0BFH
		37				108	MOV BCOUNT,A
000C 23D7		38	MOV A,#-P ; DO; TIMER=P;			109	
000E 62		39	MOV T,A	003F FF		110	
		40		0040 93		111	SEEXIT: MOV A,ATEMP
000F 55		41	SLLB: STRT T ; START TIMER			112	RETR
		42				113	
		43				114	-----
0010 6A		44	IN A,P2 ; /*CARRY=RXD*/			115	INITIALIZE ROUTINE-
		45				116	STARTS RECEIVE PROCESS
0011 F7		46	RLC A ; CARRY=P27 XOR TEST1;			117	-----
0012 5615		47	JT1 TIGRD			118	
0014 A7		48	C ; /*SHIFT CARRY INTO BUFFER*/			119	
		49				120	INIT:
		50				121	PROCEDURE;
		51	TIGRD: MOV RX0,SERBUF ; DO;			122	
0015 B020		52	SLLD: XCH A,RX0 ; DISABLE INTERRUPT;			123	
0017 20		53	RRC A ; P27=1;			124	
0018 67		54	XCH A,RX0 ; TIMER=-1;			125	
0019 20		55				126	
		56				127	
		57	DJNZ BCOUNT,SEXIT ; START EVENT COUNT;			128	
001A 043F		58				129	
		59		0041 35		130	INIT: DIS TCNT1
		60		0042 8A80		131	ORL P2,#00H
		61		0044 23FF		132	MOV A,#-1
		62		0046 62		133	MOV T,A
001C 0024		63	MOV RX0,#R0F ; END;	0047 45		134	STRT CNT
001E 27		64	CLK A ; BCOUNT=BCOUNT-1;	0048 0024		135	MOV RX0,#R0F
001F 00		65	MOV @RX0,A ; IF BCOUNT=0 THEN	004A 0001		136	MOV @RX0,#01H
0020 35		66	DJS TCNT1 ; DO; R0F=0;	004C 001E		137	MOV RX0,#1EH ; POINT AT BCOUNT
		67		004E 00C0		138	MOV @RX0,#(0C0H OR BITNO)
0021 043F		68	JMP SEXIT ; END;	0050 25		139	EN TCNT1
		69		0051 83		140	RET
		70				141	
						142	
						143	END

Figure 22. Interrupt Driven Serial Receive Program

codes such as those used in synchronous data communications (e.g. BISYNC or SDLC). BCH codes, named for their originators Bose, Chaudhuri, and Hocquenghem, are characterized by having a length of $n=2^m-1$. The number of redundant check bits can be mt where t is a positive integer (clearly $mt \leq n$). The 31,26 code fits this format with $m=5$ and $t=1$. The length of each message is $n=2^5-1=31$ with $5*1$ redundant bits, leaving 26 bits available for data transmission. With an appropriate poly-

nominal BCH codes can detect all errors consisting of $2t$ error bits and all burst errors of mt or fewer bits. The 31,26 BCH code will therefore detect any erroneous messages with 1 or 2 errors or bursts of errors of less than 5 bits. The 31,26 format (shown in Figure 23) requires the reception of a start bit followed by 31 information bits, clearly beyond the capability of the USART but perhaps within reach of a program controlled approach using the MCS-48 itself.



Figure 23. 31,26 BCH Code

A concept which reduces sensitivity to frequency deviations and thus allows the reception of longer codes is shown pictorially in Figure 24. The first line of this timing chart shows an alternative ones and zeros pattern on the RxD with a period of 5 milliseconds. The second line shows that by sampling at a period of exactly 5 milliseconds the data can be properly interpreted. The third and fourth lines show the effects of sampling with a period of six and four milliseconds respectively. In either case, an error occurs at the third sample where both periods result in sampling on an edge of the RxD signal. The third line of Figure 24 shows a hybrid sampling scheme which, based on some additional information, switches sampling periods between the two values. As can be seen in Figure 24, the data is sampled with a 4 millisecond period until the sampling begins to fall behind the data; at this point the sampling period is increased to six milliseconds and the sampling first catches up and then passes the center point of the data. As soon as this happens, the sampling period reverts to the 4 millisecond period and the cycle repeats. It can be seen that this scheme sets up a pattern which repeats indefinitely and the data can be successfully sampled. Note that the sampling pattern established is alternating periods of four and six milliseconds. The average period of this pattern, as might be expected, is 5 msec. Line 5 of Figure 24 shows the effect of a change in transmission speed to a period of 5.5 msec with no change in the sampling time. The sampling is again successful but the new sampling pattern is 4-6-6-6; 4-6-6-6, etc. Note that the average sample is again equal to the period of the received data (5.5). While this scheme

does seem to work, the question of what additional information is needed remains.

The MCS-48 must somehow decide when it is drifting out of synchronization and take corrective action. By referring back to Figure 24 it can be seen that if the MCS-48 could determine where the edges of RxD occurred with respect to its sampling times then the additional information would be available. As can be seen in the figure the choice of sampling period can be based on the following rule:

If an edge on the RxD line occurs during the first half of the current sampling period, then use the short period for the next sample. If an edge occurs during the second half of the period, then use the long sampling period for the next sample.

If the data on the RxD line does not change, of course, the MCS-48 will drift out of synchronization just as the original algorithm did. As long as edges occur on TxD, however, synchronization can be maintained. To maximize the allowable time between edges, the following addition could be made to the above rule:

If no edge occurs on the RxD line during a sample, then change sampling period from short to long or vice versa.

Note that this addition to the rule will result in using an average of the two sampling periods when no edge occurs for several bit times.

The edges of RxD can be easily detected by the use of the same structure (the Exclusive – NOR gate) which was added to the MCS-48 in Figure 20. This gate, which is used to detect the edge on RxD which begins the START bit, can naturally be used to detect any edge. Since the timer is being used to time the bit period, however, the event count input (T1) is not useful during the receive itself. By connecting the output of this gate, however, to the INT input to the MCS-48 (see Figure 25) it is possible to detect edges on RxD with the event counter when the program is trying to detect the START bit and by the external interrupt when the program is using the timer to control the sampling times.

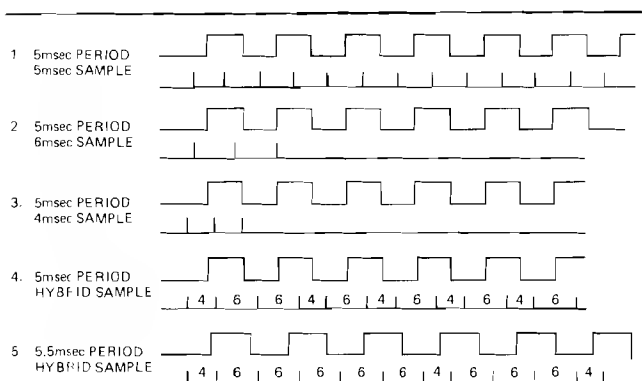


Figure 24. Various Sampling Alternatives

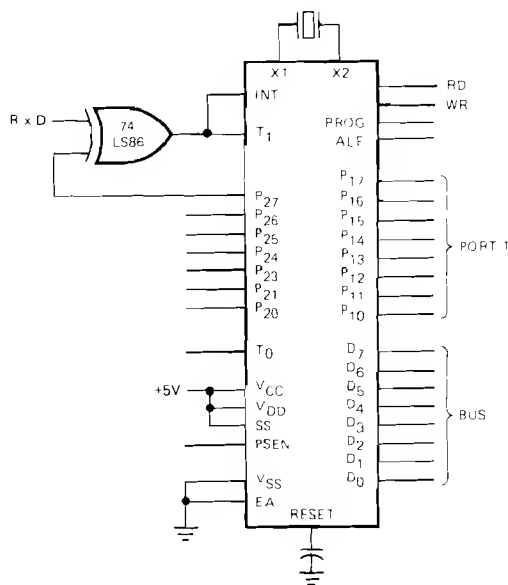
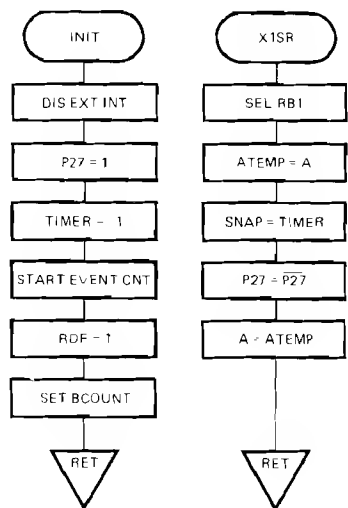


Figure 25. Modified Edge Detection

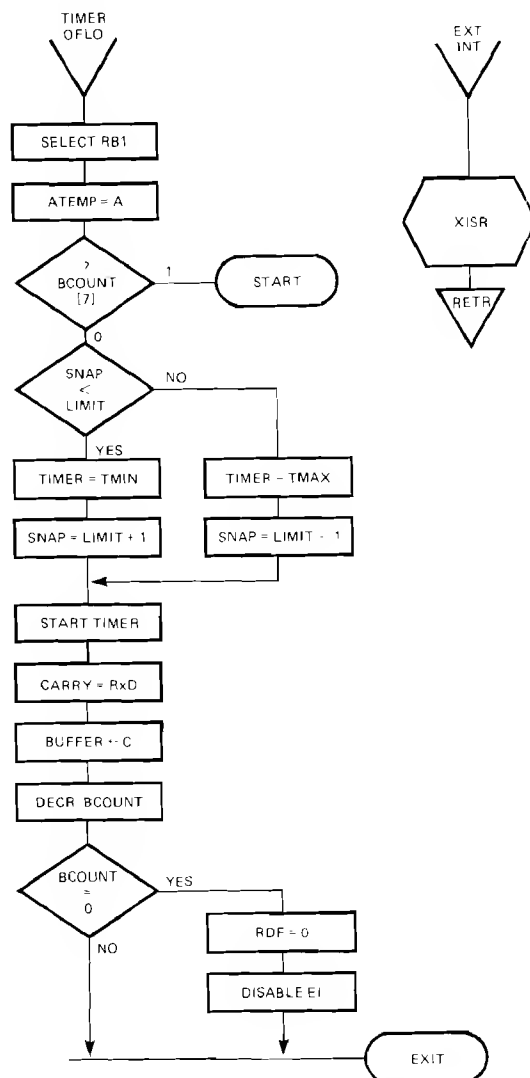
A modification to the program of Figure 21 which implements this new sampling algorithm is shown in Figure 26. The first deviation from the original program is the addition of a routine (XISR, Figure 26a which is called when an external interrupt occurs (i.e. when an edge occurs on RxD). This routine saves the status of the running program and then stores the current value of the timer register in a location called SNAP (R5 of RB1). After doing these operations the program complements bit 7 of port 2. Manipulating P27 in this manner will cause the Exclusive NOR gate to turn off the external interrupt and will set it up to generate another interrupt when the RxD line changes again (has another edge).



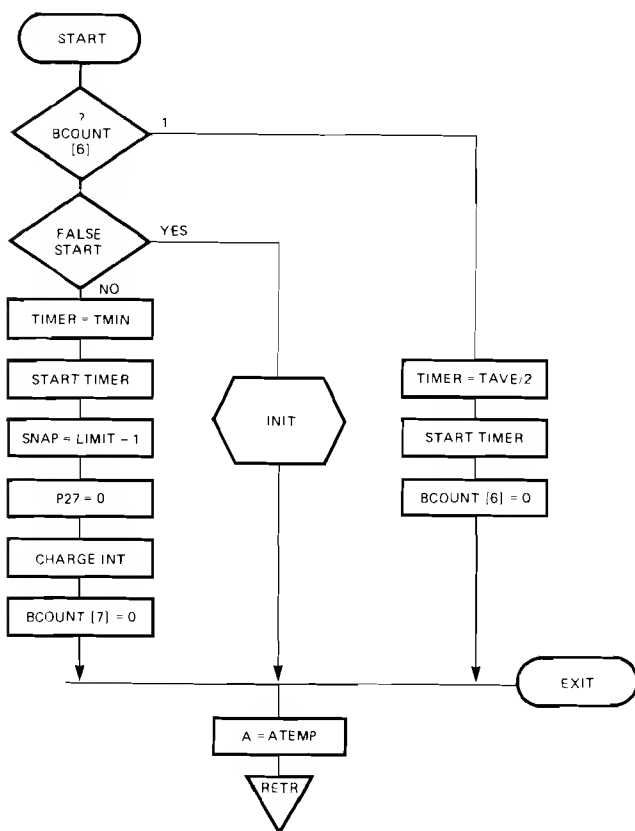
Hybrid Sampling Flowchart

Because of this edge detection it is important to condition RxD with hardware filters to ensure that the edges of RxD are clean. Any ringing will cause repeated CALLs to XISR and probable erroneous operation. The changes to the START process (Figure 26c) are two-fold; first the TIMER is set to one half the average of the two sample periods when the START bit is first detected (BCOUNT [6] = 1), and second the processing of the edge information is initialized by presetting SNAP and clearing P27.

SNAP is preset so that when the reception of data actually begins (Figure 26b BCOUNT [7] = 0), the decision block which tests SNAP against LIMIT will be initialized. This block actually compares the value in SNAP with a LIMIT value which is used to determine if the sampling point is ahead or behind the actual midpoint of the serial data. If the sampling is ahead then the timer is set for TMIN; if the sampling is behind then the timer is set for



Hybrid Sampling Flowchart



Hybrid Sampling Flowchart

TMAX. By presetting SNAP in the manner shown in the flowcharts the second rule of the algorithm, (if no edge appears on the RxD line during a sample, then change the sampling periods short to long or vice versa) is automatically met. If an edge occurs then XISR will modify SNAP, if XISR is not invoked between two samples then the choice of timer periods will alternate. The only other significant change to the algorithm is that the INIT routine must now lock out all interrupts, not just the timer overflow interrupt, while it is operating. A program which uses this algorithm to receive a 32 bit message is shown in Figure 27.

LOC	GBJ	SEQ	SOURCE STATEMENT
0		0	
1		1	*****
2		2	
3		3	SERIAL INPUT USING MCS-48
4		4	THIS CODE ASSUMES HARDWARE
5		5	SHOWN IN FIG 25. PROGRAM
6		6	IS SIMILAR TO PREVIOUS
7		7	ONE. A MORE SOPHISTICATED
8		8	SAMPLING ALGORITHM IS USED
9		9	
10		10	NOTE: A PL/M LIKE LANGUAGE WAS USED
11		11	TO COMMENT THIS LISTING AND
12		12	SEVERAL OTHERS IN THIS NOTE. NO
13		13	COMPILER EXISTS FOR THE MCS-48.
14		14	THE COMMENTS WERE 'HAND
15		15	COMPILED' INTO ASSEMBLY CODE
16		16	
17		17	*****
18		18	
19		19	-----
20		20	EQUATES
21		21	-----
22		22	
0007		23	ATEMP EQU R7 ; STORAGE FOR A DURING INTERRUPT
0006		24	BCOUNT EQU R6 ; CONTAINS NUMBER OF BITS IN MSG
0005		25	SNAP EQU R5 ; TAKES TIMER SNAP SHOT ON RxD EDGE
0002		26	COUNT EQU R2 ; UTILITY COUNTER
0000		27	Rx0 EQU R0 ; POINTER
0020		28	BITNO EQU 32 ; NUMBER OF BITS
0014		29	LIMIT EQU 20 ; TEST VALUE FOR MIN/MAX SAMPLING
FFD5		30	TMAX EQU -43 ; MAX SAMPLE PERIOD
FFD9		31	TMIN EQU -39 ; MINIMUM SAMPLE PERIOD
FFEC		32	HALF EQU -20 ; HALF NOMINAL PERIOD
0020		33	SHRBUF EQU 20H ; START OF SERIAL BUFFER
0024		34	RDF EQU 24H ; RECEIVE DONE FLAG
		35	
		36	-----
		37	CONTROL PASSED HERE ON EXT. INT.
		38	-----
0003		40	ORG 03H ; CALL SERVICE ROUTINE
0003 1466		42	ELIVEC: CALL XISR
0005 93		43	RETR
		44	
		45	-----
		46	CONTROL PASSED HERE WHEN TIMER OFLO OCCURS
		47	-----
		48	
		49	/*ENTER INTERRUPT MODE*/
0006 15		50	TMVEC: SEL RBL
0007 AF		51	MOV ATEMP,A
0008 FF		52	MOV A,BCOUNT ; IF BCOUNT[7]=0 THEN
0009 F236		53	JB7 START
		54	DO;
		55	IF SNAP<LIMIT THEN
000B FD		56	MOV A,SNAP
000C 0314		57	ADD A,#LIMIT
000E F217		58	JB7 SLIA
		59	DO;
		60	TIMER=TMIN;
		61	SNAP=LIMIT+1;
		62	END;
0010 23D9		63	MOV A,TMIN
0012 62		64	MOV T,A
0013 8D13		65	MOV SNAP,LIMIT-1
0015 041C		66	JMP SLIB
		67	ELSE
		68	DO;
		69	TIMER=TMAX;
		70	SNAP=LIMIT-1;
		71	END;
0017 2305		72	SLIA: MOV A,#TMAX
		73	

Figure 27. Hybrid Sampling Program

LOC	OBJ	SEQ	SOURCE STATEMENT	LOC	OBJ	SEQ	SOURCE STATEMENT
0019 62		74	MOV T,A	004A 1456		143	SLLD: CALL INIT
001A 6D13		75	MOV SNAP,#LIMIT-1			144	; ELSE
		76	; START TIMER;			145	; DO:
001C 55		77	SLLB: STIRI T			146	TIMER=(TMIN+TMAX)/2;
		78	; /*CARRY=RXD*/			147	START TIMER;
		79	; CARRY=P27 XOR TEST1;			148	BCOUNT[6]=0;
001L 6A		80	IN A,P2			149	END;
001E F7		81	RJC A	004C 236C		150	SLLC: MOV A,#HALF
001F 4622		82	JN11 115ND	004E 62		151	MOV T,A
0021 A7		83	CPL C	004F 55		152	STRT T
		84	; /*SHIFT CARRY INTO BUFFER*/	0050 FE		153	MOV A,BCOUNT
		85	; RX0=SERBUF;	0051 53BF		154	ANL A,#0BFH
		86	; COUNT=4;	0053 AE		155	MOV BCOUNT,A
		87	; DO WHILE COUNT<>0;			156	; END;
		88	; KSHFT MEM(RX0);			157	; /*EXIT INTERRUPT MODE*/
		89	; RX0=RX0+1;	0054 FF		158	SEXIT: MOV A,ATEMP
		90	; COUNT=COUNT-1;	0055 93		159	RETR
		91	; END;			160	
0022 6820		92	115ND: MOV RX0,#SERBUF			161	-----
0024 6A04		93	MOV COUNT,#4			162	INITIALIZE ROUTINE-
0026 26		94	SLOOP: XCH A,RX0			163	STARTS RECEIVE PROCESS
0027 67		95	RRC A			164	-----
0028 20		96	XCH A,#RX0			165	
0029 18		97	INC RX0			166	; INIT:
002A EA26		98	DJNZ COUNT,SLOOP			167	PROCEDURE;
		99	; BCOUNT=BCOUNT-1;			168	DO;
		100	; IF BCOUNT=0 THEN			169	DISABLE INTERRUPTS;
002C EE54		101	DJNZ BCOUNT,SEXIT			170	P27=1;
		102	; DO;			171	TIMER=-1;
		103	; RDF=0;			172	START EVENT COUNT;
		104	; DISABLE EX INT;			173	RDF=1;
		105	; END;			174	BCOUNT=BC0H OR BITNO
002E B824		106	MOV RX0,#RDF			175	END;
0030 27		107	CLR A			176	END INIT;
0031 A0		108	MOV #RX0,A	0056 15		177	INIT: DIS I
0032 35		109	DIS TCNTI	0057 35		178	DIS TCNTI
0033 15		110	DIS I	0058 B8B0		179	ORL P2,#80H
		111	; END;	005A 23FF		180	MOV A,#-1
0034 0454		112	JMP EXIT	005C 62		181	MOV T,A
		113	; ELSE	005D 45		182	STRT CNT
		114	; DO;	005E B824		183	MOV RX0,#RDF
		115	; IF BCOUNT[6]=0 THEN	0060 F9		184	MOV A,01
0036 FE		116	START: MOV A,BCOUNT	0061 A0		185	MOV #RX0,A
0037 D24C		117	JNB SLLC	0062 25		186	EN TCNTI
		118	; DO;	0063 BEE0		187	MOV BCOUNT,#0C0H OR BITNO
		119	; IF TEST1=0 THEN	0065 83		188	RET
0039 564A		120	JTI SLLD			189	
		121	; DO;			190	
		122	; TIMER=TMIN;			191	-----
		123	; START TIMER;			192	INTERUPT SERVICE ROUTINE
		124	; SNAP=LIMIT+1;			193	-----
		125	; P27=0;			194	; XISR:
		126	; EN I			195	PROCEDURE;
		127	; BCOUNT[7]=0;			196	DO;
		128	; END;			197	/*ENTER INTERRUPT MODE*/
003B 23D9		129	MOV A,#TMIN			198	SNAP=TIMER;
003D 62		130	MOV T,A			199	P27=NOT P27;
003E 55		131	STRT T			200	END XISR;
003F BD15		132	MOV SNAP,#LIMIT+1	0066 D5		201	XISR: SEL RBL
0041 9A7F		133	ANL P2,#7FH	0067 AF		202	MOV ATEMP,A
0043 05		134	EN I	0068 42		203	MOV A,T
0044 FE		135	MOV A,BCOUNT	0069 AD		204	MOV SNAP,A
0045 537F		136	ANL A,#7FH	006A 0A		205	IN A,P2
0047 AE		137	MOV BCOUNT,A	006B D300		206	XRL A,#00H
0048 0454		138	JMP EXIT	006D 3A		207	OUTL P2,A
		139	; ELSE	006E FF		208	MOV A,ATEMP
		140	; DO;	006F 83		209	RET
		141	; CALL INIT;			210	
		142	; END;			211	END
							; END OF PROGRAM

Figure 27. Hybrid Sampling Program

TRANSMITTING SERIAL CODE

Serial transmission is conceptually far simpler than serial reception since no synchronization is required. All that is required is to use the timer to generate interrupts at the bit rate and present the character to be transmitted serially at an I/O pin. A program which does this is shown in Figure 28. The transmission of serial data becomes much more complicated if it must occur simultaneously with reception.

If both reception and transmission are to occur simultaneously then obviously contention will exist for the use of the timer. It is possible to allow the simultaneous reception and transmission of serial data using the timer as a general clock which controls software maintained timers. The attainable baud rates using such techniques are, however, limited and the use of a 8251 USART is probably

indicated in all but the most cost sensitive applications. An exception to this rule occurs when the system, although full duplex in nature, actually transmits the same data as it receives. An example of this is a microprocessor driving a terminal such as a Teletype. Although the circuit to the terminal is full duplex, the data that is transmitted is generally the same as that received. A minor modification to the program shown in Figure 26 would implement this mode of operation. The modification would be to the XISR routine and it would add the code necessary to place the TxD I/O pin in the same state as the RxD line. Since any change in RxD results in a call to XISR, this modification would cause the retransmission of any received data. Whenever it becomes necessary to transmit data which is not being received, the program of Figure 28 could be used in a half duplex manner.

LOC	OBJ	SOURCE STATEMENT	LOC	OBJ	SOURCE STATEMENT
0			37	IN	A,P2
1 ;		-----	38	XRL	A,#00H
2 ; SERIAL TRANSMIT ON THE MC648			39	OUTL	P2,A
3 ; TO USE PUT A CHAR IN BUFF AND			40	JC	BITON
4 ; SET CHARAV TO 0FFH, WHEN THE			41	ANL	P2,#CBIT
5 ; TRANSMITTER IS READY FOR ANOTHER			42	JMP	EXIT
6 ; CHAR IT WILL CLEAR CHARAV. THE			43 BITON:	ORL	P2,#SBIT
7 ; TRANSMISSION IS DOUBLE BUFFERED.			44 EXIT:	MOV	A,ATEMP
8 ;		-----	45	RETR	
9			46		
10 ;		-----	47 ;		-----
11 ; EQUATES			48 ; BIT ROUTINE		
12 ;			49 ; -PICKS THE NEXT BIT TO TRANSMIT		
13			50 ;		-----
14 ATEMP EQU R7 ; STORAGE FOR A DURING INT.			51		
15 PTOS EQU R6 ; PARALLEL TO SERIAL CONVERTER			52 BIT:	MOV	A,COUNT
16 BUFF EQU R5 ; CHARACTER BUFFER			53	JZ	IDLE
17 CHARAV EQU R4 ; CHARACTER AVAILABLE FLAG			54	MOV	A,PTOS
18 COUNT EQU R3 ; BIT COUNTER			55	RLC	A
19 CBIT EQU 0FFH ; MASK TO CLEAR TXD IN P24			56	ORL	A,#00H
20 SBIT EQU 010H ; MASK TO SET TXD IN P24			57	MOV	PTOS,A
21 T EQU -41 ; PERIOD OF TXD			58	DEC	COUNT
22			59	RET	
23 ;		-----	60		
24 ; CONTROL PASSES HERE ON TIMER OVERFLOW			61 IDLE:	CLR	C
25 ;		-----	62	MOV	A,CHARAV
26	ORG	07H ; ENTER INTERRUPT MODE	63	JNZ	GUTONE
27			64	CPL	C
28 TOFL: SEE R81			65	RET	
29 MOV ATEMP,A			66		
30 ; SET TIMER FOR T			67 GUTONE:	MOV	A,BUFF
31 MOV A,#P			68	MOV	PTOS,A
32 MOV P,A			69	MOV	COUNT,#10
33 STIR T			70	MOV	CHARAV,#0
34 ; GET BIT INTO CARRY			71	RET	
35 CALL BIT			72		
36 ; SET TXD TO CARRY			73	END	; END OF PROGRAM

Figure 28. Serial Transmission

GENERATING PARITY

Many communications schemes require the generation and checking of parity. If a USART is used it can be programmed to automatically generate and check parity. If the communications is handled by software within the MCS-48™ then the program must perform parity calculations. Calculating parity is easy if one remembers what parity really means. A character has even parity if the number of one bits in it is even. A character has odd parity if it has an odd number of ones. The program segment shown in Figure 29 can be caused to calculate parity. It starts by setting a loop count to eight and

CONCLUSION

This Application Note has presented a very small sampling of the application techniques possible with the MCS-48™ family. The application of this new single chip computer system to tasks which have not yet yielded to the power of the micro-processor will present a fascinating challenge to the system designer.

LOC	OBJ	SEQ	SOURCE STATEMENT
		0	
		1	
		2	*****
		3	
		4	PARITY
		5	THIS PROGRAM GENERATES PARITY
		6	ON THE ACCUMULATOR
		7	CARRY WILL BE SET IF A HAS ODD PARITY
		8	
		9	*****
		10	
		11	
		12	-----
		13	EQUATES
		14	-----
		15	
0062		16	COUNT EQU R2
		17	
0100		18	PAR: ORG 100H
0100 6A8B		19	MOV COUNT,#8 ; SET LOOP COUNT
0102 97		20	CLR C ; INITIALIZE CARRY
		21	
		22	
0103 77		23	LOOP: RR A ; FOR EACH ZERO BIT IN A
0104 1207		24	JNB OVER ; COMPLEMENT THE CARRY FLAG
0106 A7		25	CPL C
0107 6A03		26	OVER: DJNZ COUNT,LOOP
		27	
		28	END ; END OF PROGRAM

Figure 29. Parity Generation

clearing the CARRY flag. After this initialization a loop is executed eight times. During each execution the accumulator is rotated and the least significant bit is tested. If the bit is a zero the CARRY flag is complemented, if the bit is a one no further action is taken. Since an even number of zeros implies an even number of ones for an eight bit character, after all eight loops have been accomplished the CARRY bit will be set if an odd number of ones were encountered; it will be reset if the number were even. Since the RR instruction does not involve CARRY the net result of executing this program loop is to set CARRY if parity is odd without effecting the character in the accumulator.



3055 Bowers Avenue
Santa Clara, California 95051
Tel: (408) 246-7501
TWX: 910-338-0026
TELEX: 34-6372

MICROCOMPUTER AND MEMORY COMPONENT SALES AND MARKETING OFFICES

AUGUST 1977

U.S. AND CANADIAN SALES OFFICES

ALABAMA

Glen White Associates
7844 Horseshoe Trail
Huntsville 35802
Tel: (205) 883-9394

ARIZONA

Sales Engineering, Inc.
7226 Sietson Drive, Suite 34
Scottsdale 85252
Tel: (602) 945-5781
TWX: 910-950-1288

INTEL CORP.

8650 N. 35th Avenue
Phoenix 85021
Tel: (602) 242-7205

CALIFORNIA

Intel Corp.*
990 E. Arques Ave.
Suite 112
Sunnyvale 94086
Tel: (408) 738-3870
TWX: 910-339-9279
TWX: 910-338-0255

Mac-I

P.O. Box 1420
Cupertino 95014
Tel: (408) 257-9880
Earle Associates, Inc.
4805 Mercury Street
Suite L
San Diego 92111
Tel: (714) 278-5441
TWX: 910-335-1585

Mac-I

P.O. Box 8763
Fountain Valley 92708
Tel: (714) 639-3341

INTEL CORP.

1651 East 4th Street
Suite 228
Santa Ana 92701
Tel: (714) 835-9642
TWX: 910-595-1114

COLORADO

Intel Corp.
12075 East 45th Avenue
Suite 310
Denver 80239
Tel: (303) 373-4920
TWX: 910-932-0322

EUROPEAN MARKETING OFFICES

BELGIUM

Intel International*
Rue du Moulin à Papier
51-Bolie 1
B-1160 Brussels
Tel: (02) 660 30 10
TELEX: 24814

ORIENT MARKETING OFFICES

JAPAN

Intel Japan Corporation*
Flower Hill-Shinmachi East Bldg.
1-23-9, Shinmachi, Setagaya-ku
Tokyo 154
Tel: (03) 426-9261
TELEX: 781-28426

INTERNATIONAL DISTRIBUTORS

ARGENTINA

S.E. SA
Av. Pío Rogue Saenz Pena 1142 9B
1035 Buenos Aires
Tel: 35-6784

AUSTRALIA

A. J. Ferguson (Adelaide) PTY. Ltd.
44 Prospect Rd.
Prospect 5082
South Australia 17005
Tel: 269 1244
TELEX: 82635

A. J. Ferguson Electronics

34 Herbert Street
West Ryde, N.S.W. 2114
Tel: AC8 269 1244
TELEX: 82635

Warburton-Frankie (Sydney) Pty. Ltd.
199 Parramatta Road
Auburn, N.S.W. 2114
Tel: 648-1711, 648-1381
TELEX: WARFRAN AA 22265
Warburton-Frankie Industries
(Melbourne) Pty. Ltd.
220 Park Street
South Melbourne, Victoria 3205

AUSTRIA

Bacher Elektronische Geräte GmbH
Meidlinger Hauptstrasse 78
A 1120 Vienna
Tel: (0222) 83 63 96
TELEX: (01) 1532

BELGIUM

Inelco Belgium S.A.
Avenue Val Duchesse, 3
B-1160 Brussels
Tel: (02) 660 00 12
TELEX: 25441

CONNECTICUT

Intel Corp.
Peacock Alley
1 Padanaram Road
Danbury 06810
Tel: (203) 792-8366

FLORIDA

Intel Corp.
2020 W. McNab Road, Suite 104
Ft. Lauderdale 33309
Tel: (305) 971-7200
TWX: 510-956-9407
Intel Corp.
5151 Adanson Street, Suite 105
Orlando 32804
Tel: (305) 628-2293
TWX: 810-853-9219

ILLINOIS

Intel Corp.*
1000 Jorie Boulevard
Suite 224
Oakbrook 60521
Tel: (312) 325-9510
TWX: 910-651-5881

IOWA

Technical Representatives, Inc.
1733 Hillside Drive N.W.
Cedar Rapids 52405
Tel: (319) 396-5662

KANSAS

Technical Representatives, Inc.
801 Clairborne
Olathe 66061
Tel: (913) 782-1177
TWX: 910-748-6412

MARYLAND

Glen White Associates
57 West Timonium Road
Timonium 21093
Tel: (301) 252-6360
Intel Corp.*
57 West Timonium Road
Suite 307
Timonium 21093
Tel: (301) 252-7742
TWX: 710-232-1807

FRANCE

Intel Corporation, S.A.R.L.*
74, Rue d'Arcueil
Siliç 223
94528 Rungis Cedex
Tel: (01) 687 22 21
TELEX: 270475

TAIWAN

Taiwan Automation Co.*
6th Floor, 18-1 Lane 14
Chi-Lin Road
Taipei
Tel: (02) 551726-9
TELEX: 11942 TAIAUTO

DENMARK

Scandinavian Semiconductor
Supply A/S
Nannasgade 18
DK-2200 Copenhagen N
Tel: (01) 93 50 90
TELEX: 19037

FINLAND

Oy Finronic AB
Loennrotinkatu 35D
SF 00180
Helsinki 18
Tel: (90) 664 451
TELEX: 12426

ISRAEL

Tekelec Airtronic
Cite des Bruyeres
Rue Carle Vernet
19930 Sevres
Tel: (1) 027 75 35
TELEX: 250997

GERMANY

Allred Neye Enelachnik GmbH
Schillerstrasse 14
D-2085 Quickborn-Hamburg
Tel: (04106) 8121
TELEX: 02-13590
Electronic 2000 Vertriebs GmbH
Neumarkter Strasse 75
D-8000 Muenchen 80
Tel: (089) 41 34061
TELEX: 522561
Jermyn GmbH
Postfach 1146
D-6277 Kamberg
Tel: (06434) 6005
TELEX: 484428

MASSACHUSETTS

Intel Corp.
187 Billerica Road, Suite 14A
Chelmsford 01824
Tel: (617) 256-6567
TWX: 710-343-6333

MICHIGAN

Intel Corp.
26500 Northwestern Hwy
Suite 401
Southfield 48075
Tel: (313) 353-0920
TWX: 910-420-1212
TELEX: 231143

MINNESOTA

Intel Corp.
8200 Normandale Avenue
Suite 422
Bloomington 55437
Tel: (612) 835-6722
TWX: 910-576-2867

MISSOURI

Technical Representatives, Inc.
Trade Center Bldg
300 Brookside Drive, Suite 108
Hazelwood 63042
Tel: (314) 731-5200
TWX: 910-762-0618

NEW JERSEY

Intel Corp.
2 Kilmer Road
Edison 08817
Tel: (201) 985-9100
TWX: 710-480-6238

NEW YORK

Intel Corp.*
350 Vanderbilt Motor Pkwy.
Suite 402
Hempstead 11757
Tel: (516) 231-3300
TWX: 510-221-2198
Intel Corp.
474 Thurston Road
Rochester, N.Y. 14619
Tel: (716) 329-7340
TWX: 510-253-3841

SCANDINAVIA

Intel Scandinavia A/S*
Lyngbyvej 32 2nd Floor
DK-2150 Copenhagen East
Denmark
Tel: (01) 18 20 00
TELEX: 19567
Intel Sweden AB*
Box 20092
S-16120 Bromma
Sweden
Tel: (08) 98 53 90
TELEX: 12261

HONG KONG

ASTEC International
Oriental Centre
14th Floor, No. 67-71
Chatham Road
Kowloon, Hong Kong
Tel: 3-694751
Cable: "ASCOMP"
TELEX: 74899 ASCOM HX

INDIA

Electronics International
128 Mahatma Gandhi Road
Secunderabad
Tel: 53211
TELEX: 043-227

ISRAEL

Electronics Ltd.*
11 Rozans Street
P.O. Box 39306
Tel-Aviv
Tel: 475151
TELEX: 33638

ITALY

Eledra 3S S.P.A.*
Viale Elvezia, 18
20150 Milan
Tel: (02) 3493041
TELEX: 29332
Eledra 3S S.P.A.*
Via Paolo Gaidano, 141 D
10137 Torino
TEL: (011) 30 57 097 - 30 97 114
Eledra 3S S.P.A.*
Via Giuseppe Valmarana, 62
00139 Rome, Italy
Tel: (06) 81 27 290 - 81 27 324
TELEX: 63051

JAPAN

Pan Electron
No. 1 Higashikata-Machi
Midori-Ku, Yokohama 226
Tel: (045) 471 8811
TELEX: 781-4 73

NEW YORK (cont.)

T-Squared
4054 Newcourt Ave.
Syracuse 13206
Tel: (315) 463-8592
TWX: 710-541-0554
T-Squared
640 Kragg Rd.
P.O. Box W
Pittsford 14534
Tel: (716) 381-2551
TELEX: 97-8289
Intel Corp.
85 Market Street
Poughkeepsie, New York 12601
Tel: (914) 473-2303
TWX: 510-248-0060
NORTH CAROLINA
Glen White Associates
3700 Computer Dr., Suite 330
Raleigh 27609
Tel: (919) 737-7016

OHIO

Intel Corp.*
8312 North Main Street
Dayton 45415
Tel: (513) 890-5350
TELEX: 288-004
Intel Corp.*
26250 Euclid Ave
Suite 51F
Euclid 44132
Tel: (216) 289-0101

PENNSYLVANIA

Intel Corp.*
520 Pennsylvania Ave.
Fort Washington 19034
Tel: (215) 542-9444
TWX: 510-661-0709
TENNESSEE
Glen White Associates
Rt. 172 Norwood S.D.
Jonesboro 37659
Tel: (615) 928-0184
Glen White Associates
2523 Howard Road
Germantown 38138
Tel: (901) 754-0483

ENGLAND

Intel Corporation (U.K.) Ltd.*
Broadfield House
4 Between Towns Road
Cowley, Oxford OX4 3NB
Tel: (0865) 77 14 31
TELEX: 837203

Intel Corporation (U.K.) Ltd.

46-50 Beam Street
Nantwich, Cheshire CW5 5LJ
Tel: (0270) 62 65 60
TELEX: 36620

JAPAN (cont.)

Ryoyo Electric Corp.
Konwa Bldg.
1-12-22, Tsukiji, 1-Chome
Chuo-Ku, Tokyo 104
Tel: (03) 543-7-11
Nippon Micro Computer Co. Ltd.
Mitsum Bldg. 4-5-21 Kojimachi
Chiyoda-Ku, Tokyo 102
Tel: (03) 230-0041

KOREA

Koran Digital
Sam Yung Bldg. #903
71-2 Bukchang - Dong Chung-Ku
Seoul 100

NETHERLANDS

Inelco Nederland
AFD Elektronik
Jaan Muiskenweg 22
NL-1006 Amsterdam
Tel: (020) 934824
TELEX: 14622

NORWAY

Nordisk Elektronik (Norge) A/S
Mustads Vei 1
N-0510 2
Tel: (02) 55 38 93
TELEX: 15963
PORTUGAL
Ditram
Componentes E. Electronica LDA
Av. Miguel Bombarda 133
Lisboa 1
Tel: 119 45 313

SOUTH AFRICA

Electronic Building Elements
P.O. Box 4609
Pretoria
Tel: 78 92 21
TELEX: 30181

TEXAS

Mycrosystems Marketing Inc.
13777 N. Central Expressway
Suite 405
Dallas 75231
Tel: (214) 238-7157
TWX: 910-867-4763
Mycrosystems Marketing Inc.
5610 Harwin Avenue, Suite 125
Houston 77036
Tel: (713) 783-2900
Mycrosystems Marketing Inc.
2622 Geronimo Trail
Austin 78746
Tel: (512) 266-1750
Intel Corp.
2925 L.B.J. Freeway
Suite 100
Dallas 75234
Tel: (214) 241-9521
TWX: 910-860-5487

VIRGINIA

Glen White Associates
P.O. Box 1104
Lynchburg 24505
Tel: (804) 384-8920

WASHINGTON

E.S. Chase Co.
P.O. Box 80903
Seattle 98108
Tel: (206) 762-4824
TWX: 910-444-2298

CANADA

Intel Corp.
70 Chamberlain Ave
Ottawa, Ontario K1S 1V9
Tel: (613) 232-8576
TELEX: 053-4419
Multitek, Inc.*
4 Barran Street
Ottawa, Ontario K2J 1G2
Tel: (613) 825-4553
TELEX: 053-4585

GERMANY

Intel Semiconductor GmbH*
Siedtstrasse 27
8000 Muenchen 2
Tel: (089) 55 81 41
TELEX: 523 177
Intel Semiconductor GmbH
Abraham Lincoln Strasse 30
6200 Wiesbaden 1
Tel: (06121) 74855
TELEX: 04186183
Intel Semiconductor GmbH
D-7000 Stuttgart 80
Ernstthaldenstrasse 17
Tel: (0711) 7351506
TELEX: 7255346
Intel Semiconductor GmbH
Wiesenweg 26
D-6272 Niederhausen
Tel: (06127) 2314

SPAIN

Interface
Ronda San Pedro 22
Barcelona 10
Tel: 301 78 51

SWEDEN

Nordisk Elektronik AB
Fack
S-10380 Stockholm 7
Tel: (08) 246340
TELEX: 10547

SWITZERLAND

Inoustrade AG
Gemsenstrasse 2
Postfach 80 - 21190
CH-8021 Zurich
Tel: (01) 60 22 30
TELEX: 36788

UNITED KINGDOM

Rapid Runt
11-15 Betterton Street
Drury Lane
London WC2H 9BS
Tel: (01) 379-6741
TELEX: 28752
G.E.C. Semiconductors Ltd
East Lane
Wembley HA9 7PP
Middlesex
Tel: (01) 904-9303
TELEX: 923429
Jermyn Industries
Vestry Estate
Sevenoaks, Kent
Tel: (0732) 50144
TELEX: 95142

*Field Application Location